massachusetts institute of technology — artificial intelligence laboratory

Reinforcement Learning by Policy Search

Leonid Peshkin

AI Technical Report 2003-003 February 2003

Reinforcement Learning by Policy Search

Leonid Peshkin

B. S., Moscow Technical University (1993)
M.Sc., Weizmann Institute, Israel (1995)

Submitted to the Department of Computer Science in
partial fulfillment of the requirements for the degree of

Doctor of Philosophy, Computer Science

November 2001

Supervisor: Leslie P. Kaelbling
Professor of Computer Science, MIT

Reader: John N. Tsitsiklis
Professor of Computer Science, MIT

Reader: Thomas L. Dean
Professor of Computer Science, Brown University

Reinforcement Learning by Policy Search

by

Leonid Peshkin

One objective of artificial intelligence is to model the behavior of an intelligent agent interacting with its environment. The environment's transformations can be modeled as a Markov chain, whose state is partially observable to the agent and affected by its actions; such processes are known as partially observable Markov decision processes (POMDPs). While the environment's dynamics are assumed to obey certain rules, the agent does not know them and must learn.

In this dissertation we focus on the agent's adaptation as captured by the reinforcement learning framework. This means learning a policy—a mapping of observations into actions—based on feedback from the environment. The learning can be viewed as browsing a set of policies while evaluating them by trial through interaction with the environment.

The set of policies is constrained by the architecture of the agent's controller. POMDPs require a controller to have a memory. We investigate controllers with memory, including controllers with external memory, finite state controllers and distributed controllers for multi-agent systems. For these various controllers we work out the details of the algorithms which learn by ascending the gradient of expected cumulative reinforcement.

Building on statistical learning theory and experiment design theory, a policy evaluation algorithm is developed for the case of experience re-use. We address the question of sufficient experience for uniform convergence of policy evaluation and obtain sample complexity bounds for various estimators. Finally, we demonstrate the performance of the proposed algorithms on several domains, the most complex of which is simulated adaptive packet routing in a telecommunication network.

Keywords: POMDP, policy search, gradient methods, reinforcement learning, adaptive systems, stochastic control, adaptive behavior.

Contents

Preface

This dissertation presents work started at Brown University and completed at the Artificial Intelligence Laboratory, MIT over the course of several years. A significant part of the research presented in this dissertation has been previously published elsewhere.

The work on learning with memory, presented in Chapter 2 constitutes common work with Nicolas Meuleau and significantly overlaps with presentation in Proceedings of the Sixteenth International Conference on Machine Learning [123] and in Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence [104].

A material on cooperation in games from Chapter 3 constitutes common work with Nicolas Meuleau, Kee-Eung Kim and Leslie Kaelbling and was published in Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence [122].

Finally, the results in Chapters 6 and 5 appear in several publications. In particular in the joint work with Sayan Mukherjee, published in the proceedings of the Fourteenth Annual Conference on Computational Learning Theory [124]; joint work with Nicolas Meuleau and Kee-Eung Kim from the MIT technical report [103]; joint work with Christian Shelton [126] to be published in the proceedings of the Nineteenth International Conference on Machine Learning.

This is a large document of 134 pages which means I might have to make corrections even after it has been published. I will maintain the errata list and provide an updated copy of this document on my current internet page. The best way to locate it is to query your favorite Internet search engine with "Leon Peshkin thesis Reinforcement Learning Policy Search". Please make sure you are reading the most recent copy. Currently it is at `http://www.ai.mit.edu/~pesha/disser.html`

Acknowledgments could have taken as much space and time as the rest of the document. Without some people, this entire work would have not been possible, while others gave a brief but important support and encouragement. It would be difficult to describe everyone's contribution, to prioritize or to categorize people into mentors, comrades, influences and inspirations. Perhaps instead of the exercise in self-decomposition into elementary influenceicals I can contribute to a genre of acknowledgments by actually naming everyone to whom I am grateful.

Dear Justin Boyan, Misha & Natalia Chechelnitsky, Abram Cherkis, Tom Dean, Ikka Delamer, Abram & Sofa Dorfman, Max Eigel, John Eng-Wong, Theos Eugeniou, Mikhail Gelfand, Sergei & Galya Gelfand, Stuart Geman, Ambarish Ghosh, Matteo Golfarelli, Polina Golland, Lisa Gordon, Vladimir Gurevich, Samuel Heath, Susan Howe, Yuri Ivanov, Grigorii Kabatyansky, Erin Khaiser, Eo-Jean Kim, Kee-Eung Kim, Max Kovalenko, Serg & Rada Landar, Sonia Leach, Geesi Leier, Eugene Levin, Serg Levitin, Michael Littman, Arnaldo Maccarone, Lada Mamedova, Nicolas & Talia Meuleau, Leonid *et al.* Mirnys, Louise Mondon, Sayan Mukherjee, Luis Ortiz, Donald *et al.* Peshkins, Avi Pfeffer, Claude & Henry Presset, Yuri Pryadkin, Ida Rawnitzky, Boris & Tonia Reizis, Luba & Michael Roitman, Smil Ruhman, Virginia Savova, Michael Schwarz, Ehud Shapiro, Christian Shelton, Bill Smart, Rich Sutton, Mike Szydlo, John Tsitsiklis, Shimon Ullman, Paulina Varshavskaia, Peter Wegner, Michael Yudin: I remember and value what you have done for me.

I feel very fortunate to have worked with Leslie Kaelbling. She generously devoted knowledge and efforts when I required attention, yet patiently allowed me to pursue various interests or to wonder around for motivation. She will always be my role model for her curiosity and discipline, jolliness and seriousness, courage and compassion, principality and liberalism.

I would like to acknowledge that my parents, Miron and Claudia, have heroically accepted my choice to be away from the family in order to continue my education; and gave me all their love and support throughout years of separation. Finally, I would like to dedicate this dissertation to the memory of my grandparents: Rebeka Levintal from Sabile, Latvia; Aaron Peshkin from Slonim, Belarus; Boruh Logwinsky and Sarah Kazadoi from Uman, Ukraine.

Introduction

"The world we live in today is much more a man-made or artificial world than it is a natural world" [154]. In such a world there are many systems and environments, both real and virtual, which can be very well described by formal models. This creates an opportunity for developing a "synthetic intelligence"—artificial systems which cohabit these environments with human beings and carry out some useful function. In this work we address some aspects of this development in the framework of reinforcement learning [167]. Reinforcement learning is learning what to do—how to map sensations to actions—from feedback. In other words, how to behave so as to maximize a numerical reward signal (see figure 1). Unlike in other forms of machine learning, the learner is not told which actions to take, but rather must discover which actions yield the most reward by trying them. In some challenging cases, actions may affect not only the immediate reward, but also the next sensation and, through that, all subsequent rewards. These two characteristics—search by trial-and-error and delayed reward—are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is not defined by specifying some particular method of learning or a set of learning algorithms. Rather, any algorithm that is suited to solve a characteristic learning problem is considered a reinforcement learning algorithm. A full specification of the reinforcement learning problem in terms of optimal control of Markov decision processes is given in Chapter 1, but the basic idea is to capture the most important aspects of a learning agent interacting with its environment to achieve a goal. Clearly such an agent must be able to sense the state of the environment to some extent and to take actions that affect that state. The agent must also have a goal or goals relating to the state of the environment.

A variety of decision making and stochastic control challenges can be formulated and tackled in the context of RL. Among the successes of RL are the self-improving backgammon player [173]; a scheduler adjusting performance of multi-elevator complex [40, 41]; a space-shuttle scheduling algorithm [191] and channel allocation and control for cellular telephone systems [156]. In our opinion the most impressive application of RL algorithms to robotics is due to Dr. Hajime Kimura *et al.* for mechanical systems learning to crawl and walk [74, 75].

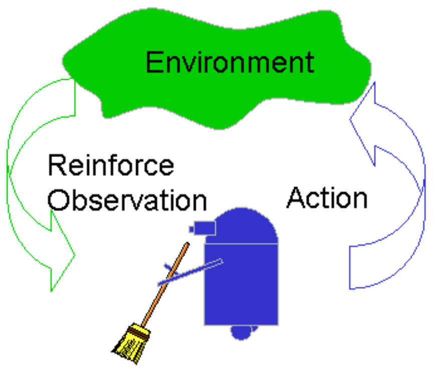


Fig. 1: The paradigm of a learning system.

Decision making is a very important aspect of intelligence. Understanding how to build decision making algorithms would not only enable new technologies, but benefit cognitive theories of natural intelligence. It is to a large extent an open question whether any reinforcement learning algorithm constitutes a biologically plausible way of learning. There are two separate issues here. One is to establish the principal possibility for a reinforcement learning algorithm to be implemented in biological hardware—in a system assembled from living cells (for discussion of related issues see [140]). Another is to identify this hardware in a living organism as well as the details of the implementation of the algorithm. The actor-critic class of algorithms [80, 79] is examined in this light, in particular by Dayan and Abbott [44](Chapter 9) and Dayan *et al.* [71, 106, 150]. Policy search algorithms considered in this dissertation also lend themselves to arguments of biological plausibility (see, e.g., [11, 125]).

The problem formulation itself is however very intuitive and relates closely to everyday practice of acting and making decisions under uncertainty. Very often there is feedback available which corresponds to how well one acts, while generally it is expected from the individual to perform poorly at initial stages, then learn from experience assuming that there are some regularities and statistical structure to the task. There is rarely time to examine all possible choices and outcomes and decide upon a sequence of actions—a plan. Commonly in stochastic environments human decision makers formulate a policy—a set of immediate initial responses to the circumstances.

It is important to get one common confusion out of the way. The general setup of the decision making under uncertainty is very similar to the one investigated for decades in the field of Operations Research. Indeed, reinforcement learning has grown out of this field and builds upon its techniques. The essential difference is that in reinforcement learning no knowledge of the environment's organization and dynamics is assumed, which turns a planning into learning problem. Littman's dissertation [88] provides a good in-depth description of relations between OR and RL. Naturally, what could already be a difficult planning problem, becomes even harder when decision making is made on-line without the environment's model at hand.

In this work we put forward the thesis that the general task of reinforcement learning stated in a traditional way seems to be unreasonably ambitious for complex domains. Different ways of leveraging information about the problem at hand are considered. We investigate general ways of breaking the task of designing a controller down to more feasible sub-tasks which are solved independently. We propose to consider both taking advantage of past experience by reusing parts of other systems, and facilitating the learning phase by employing a bias in initial configuration.

In reinforcement learning, the main challenges often consist in the need to operate under conditions of incomplete information about the environment. One case of incomplete information is when the agent does not observe all aspects of the environment, or observes some transformation of the environment state, which makes things look ambiguous. This is the case of so-called partial observability. Another case of incomplete information arises when parts of the system are controlled independently and therefore one part is not necessarily aware of other parts' decision or sensations. This is the case of distributed control. We present learning algorithms for these cases.

One set of reinforcement-learning techniques that have been applied to learning both in cooperative games and under partial observability conditions in single-agent systems are *value-based.* They rely on estimating a value or utility of occupying particular states of the environment or taking particular actions in response to being in a state. Unfortunately, application of these techniques is only justified when the environment state is completely observable to the agents and therefore the notion of cumulative reinforcement called *value* makes sense.

Policy search methods are a reasonable alternative to value-based methods for the case of partially observable environments. The general idea behind these methods is to search for optima in space of all possible policies (behaviors) by directly examining different policy parameterizations, bypassing the assignment of the value. There is no general way to solve this global optimization problem, so our only option is to explore different approaches to finding local optima.

Facing the dilemma of having to solve hard optimization in a "global" sense, while being capable rather of doing optimization in a "local" sense both spatially and temporally is paralleled by human behavior and is present in various engineering considerations. Genetic algorithms and genetic programming constitute an example of direct policy search methods, but are completely outside of the scope of research presented here. In this work we focus on gradient-based methods for finding local optima. In brief, we perform a stochastic gradient ascent in the policy-parameter space, by moving along the direction of the steepest ascent of the "goodness" function, which is being estimated through experience. This principle developed in a variety of controller architectures turns out to be surprisingly successful. Ultimately, each learning algorithm presented in this dissertation consists of evaluating the current course of actions by trial and error, assigning some credit to every part of the controller's mechanism for what was experienced, making an adjustment and reevaluating. This natural learning procedure turns out to have a rigorous mathematical justification.

Organization

The rest of this text is organized as follows. Chapter 1 presents the model of sequential decision making and establishes the notation. A brief introduction to the field of reinforcement learning consists of the definitions of Markov decision processes with complete and partial observability and other concepts necessary for the setup of reinforcement learning. We consider the notion of optimality and ways of learning optimal decision strategies, which depend on the concept of value attributed to a particular episode. Methods applicable for the completely observable case turn out not to extend well for the partially observable setting. In this setting, a learning problem could be formulated and solved as a stochastic optimization problem or a policy search problem. Policy search methods are the focus of this dissertation. The chapter concludes with an overview of policy search methods in general and related work on gradient methods in policy search in particular.

In order for an agent to perform well in partially observable domains, it is usually necessary for actions to depend on the history of observations. Therefore, optimal control requires the use of memory to store some information about the past. Chapter 2 describes how to build a controller with memory. It begins by developing in detail the gradient ascent algorithm for the case of memoryless policies. This simple controller is combined with memory in various ways. First a *stigmergic* approach is explored, in which the agent's actions include the ability to set and clear bits in an external memory, and the external memory is included as part of the input to the agent. Then, the algorithm for the case of finite state controllers is developed, in which memory is a part of the controller. The advantages of both architectures are illustrated and performance is contrasted with other existing approaches on empirical results for several domains.

At this point the discussion turns to a multi-agent setting. Chapter 3 examines the extension of previously introduced gradient-based algorithms to learning in cooperative games. Cooperative games are those in which all agents share the same payoff structure. For such a setting, there is a close correspondence between learning in a centrally controlled distributed system and in a system where components are controlled separately. A resulting policy learned by the distributed policy-search method for cooperative games is analyzed from the standpoint of both local optimum and Nash equilibrium—game-theoretic notions of optimality for strategies. The effectiveness of distributed learning is demonstrated empirically in a small, partially observable simulated soccer domain.

In chapter 4 we validate the RL algorithm developed earlier in a complex domain of network routing. Successful telecommunications require efficient resource allocation which can be achieved by developing adaptive control policies. Reinforcement learning presents a natural framework for the development of such policies by trial and error in the process of interaction with the environment. Effective network routing means selecting the optimal communication paths. It can be modeled as a multi-agent RL problem and solved using the distributed gradient ascent algorithm. Performance of this method is compared to that of other algorithms widely accepted in the field. Conditions in which our method is superior are presented.

Stochastic optimization algorithms used in reinforcement learning rely on estimates of the value of a policy. Typically, the value of a policy is estimated from results of simulating that very policy in the environment. This approach requires a large amount of simulation as different points in the policy space are considered. In chapter 5, we develop value estimators that use data gathered when using one policy to estimate the value of using another policy, for some domains resulting in much more data-efficient algorithms.

Chapter 6 addresses the question of accumulating sufficient experience for uniform convergence of policy evaluation as related to various parameters of environment and controller. We derive sample complexity bounds analogous to these used in statistical learning theory for the case of supervised learning. Finally, chapter 7 summarizes the work presented in this dissertation. It draws conclusions, outlines contributions and suggests several directions for further development of this research.

Chapter 1

Reinforcement Learning

Summary This chapter presents the model of sequential decision making and establishes the notation. A brief introduction to the field of reinforcement learning consists of the definitions of Markov decision processes with complete and partial observability and other concepts necessary for the setup of reinforcement learning. We consider the notion of optimality and ways of learning optimal decision strategies, which depend on the concept of value attributed to a particular episode. Methods applicable for the completely observable case turn out not to extend well for the partially observable setting. In this setting, a learning problem could be formulated and solved as a stochastic optimization problem or a policy search problem. Policy search methods are the focus of this dissertation. The chapter concludes with an overview of policy search methods in general and related work on gradient methods in policy search in particular.

1.1 Markov Decision Processes

Reinforcement learning is the process of learning to behave optimally with respect to some scalar feedback value over a period of time. The learning system does not get to know the correct behavior, or the true model of the environment it interacts with. Once given the sensation of the environment state $s(t)$ at time t as an input (see figure 1.1), the agent chooses the action $a(t)$ according to some rule,

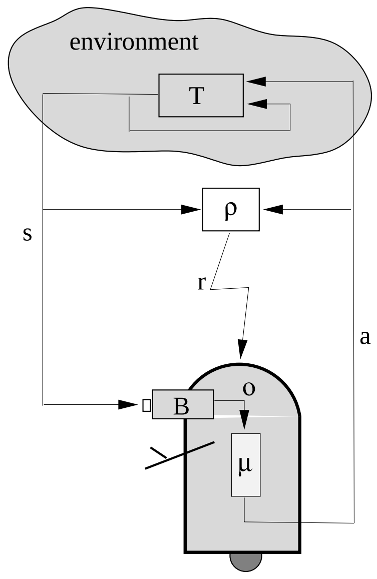


Fig. 1.1: The architecture of a learning system.

often called a *policy*, denoted μ. This action constitutes the output. The effectiveness of the action taken and its effect on the environment is communicated to the agent through a scalar value $r(t)$, called the *reinforcement signal*, sometimes described as *reward*, *cost* or *feedback*.

The environment undergoes some transformation described by the process $\mathcal{T}$ and changes current state $s(t)$ into the new state $s(t{+}1)$. A few important assumptions about the environment are made. In particular, the so-called Markov property is assumed: given the most recent events, the next state is independent of the history. Usually we assume a *non-deterministic* environment, which means that taking the same action in the same state could lead to a different next state and generate different feedback signal. Also, mostly for the purpose of the theoretical analysis of learning algorithms, we assume that the environment is *stationary*: i.e., that the probabilities of the next state and reinforcement, given the current state and action, do not change with time.

MDP The class of problems described above can be modeled as Markov decision processes (MDPs). An MDP is a 4-tuple $\langle S, A, \mathcal{T}, \rho \rangle$, where:

- S is the set of states;
- A is the set of actions;
- $\mathcal{T}: S \times \mathcal{A} \rightarrow \mathcal{P}(S)$ is a mapping from states of the environment and

actions of the agent to probability distributions[1] over states of the environment; and

- $\rho : S \times A \to \mathcal{R}$ is the payoff function[2], mapping states of the environment and actions of the agent to immediate reward. We assume that the reward $\rho(s, a)$ is bounded by some fixed value r_{max} for any s and a.

POMDP The more complex case is when the agent is no longer able to reliably determine which state of the MDP it is currently in. This situation is sometimes called *perceptual aliasing*, since several states of the environment induce the same observation. The process of generating an observation is modeled by an observation function $B(s(t))$. The resulting model is a *partially observable Markov decision process* (POMDP). In a POMDP, at each time step (see Figure 1.2): the agent observes $o(t)$ corresponding to $B(s(t))$ and performs an action $a(t)$ according to its strategy, inducing a state transition of the environment; then receives the reward $r(t)$. Obviously, an MDP is a trivial case of a POMDP with the degenerate observation function.

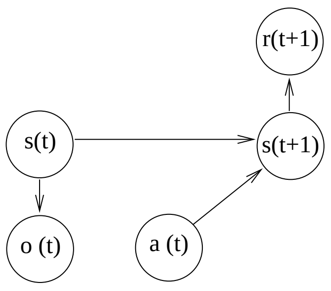


Fig. 1.2: An influence diagram for an agent in a POMDP.

Formally, a POMDP is defined as a tuple $\langle S, O, A, B, \mathcal{T}, \rho \rangle$ where:

- S is the set of states;
- O is the set of observations;
- A is the set of actions;
- B is the observation function $B : S \to \mathcal{P}(O)$;
- $\mathcal{T} : S \times \mathcal{A} \to \mathcal{P}(S)$ is a mapping from states of the environment and actions of the agent to probability distributions over states of the environment;

[1]Let $\mathcal{P}(\Omega)$ denote the set of probability distributions defined on some space Ω.
[2]it is sometimes called *reinforcement* or feedback

- $\rho : S \times A \to \mathcal{R}$ is the payoff function, mapping states of the environment and actions of the agent to immediate reward.

Experience A cycle of interaction between the agent and the environment results in a sequence of events. These events, which are observable by an agent, constitute its experience. We denote by H_t the set of all possible experiences of length t:

$$H_t = \{\langle o(1), a(1), r(1), \ldots, o(t), a(t), r(t), o(t+1)\rangle\} ,$$

where $o(t) \in O$ is the observation of the agent at time t; $a(t) \in A$ is the action the agent has chosen to take at time t; and $r(t) \in \rho$ is the reward received by the agent at time t. In order to specify that some element is a part of the experience h at time τ, we write, for example, $r(\tau, h)$ and $a(\tau, h)$ for the τ^{th} reward and action in the experience h. We will also use h^τ to denote a prefix of the sequence $h \in H_t$ truncated at time $\tau \leq t$: $h^\tau \overset{\text{def}}{=} \langle o(1), a(1), r(1), \ldots, o(\tau), a(\tau), r(\tau), o(\tau+1)\rangle$. Sometimes we would want to discuss a set of events which includes but is not limited to the experience, e.g. actual state of the environment $s(t)$. This augmented sequence of events will be called a *history*.

Return An experience $h{:}\langle r(1) \ldots r(i) \ldots\rangle$ includes several immediate rewards, that can be combined to form a *return* $R(h)$. There are different ways to quantify the optimality of the agent's behavior in the underlying Markov decision process $(S, A, \mathcal{T}, \rho)$. We present three possibilities here, and will mostly concentrate on the last one. Also, we focus on returns that may be computed (or approximated) using the first N steps, and are bounded in absolute value by R_{max}.

The simplest *finite horizon* criterion takes into account the expected reward for the next N steps to form the undiscounted finite horizon return: $R(h) = \sum_{t=0}^{N} r(t, h)$. In this case $R_{max} = Tr_{max}$. This criterion often is not appropriate, since in most cases an agent does not know the length of its life.

The *average reward* criterion considers the average reward over an infinite life span to form the undiscounted infinite horizon return:

$$R(h) = \lim_{N\to\infty} \frac{1}{T} \sum_{t=0}^{N} r(t, h) .$$

One problem with this criterion is that an agent's extremely ineffective early behavior could be overlooked due to the averaging with the long-run reward.

In the *infinite horizon discounted reward* criterion, behavior is optimized in the following way [65, 133]: the aim is to maximize the long-run reward, but rewards that are received in the future are geometrically discounted by the discount factor $\gamma \in (0, 1)$:

$$R(h) = \sum_{t=0}^{\infty} \gamma^t r(t, h) \ .$$

The discount factor γ could be interpreted as the probability of existing for another step, or as an inflation rate. It is quite intuitive to value a payoff received a year from now less than an immediate payoff even if they amount to the same quantity. In this case we can approximate R using the first $T_\epsilon = \log_\gamma \frac{\epsilon}{R_{max}}$ immediate rewards. Using T_ϵ steps we can approximate R within ϵ since $R_{max} = \frac{r_{max}}{1-\gamma}$ and $\sum_{t=0}^{\infty} \gamma^t r(t) - \sum_{t=0}^{T_\epsilon} \gamma^t r(t) < \epsilon$. It is important that we are able to approximate the return in T steps, since the length of the horizon comes up as an important parameter in our calculations.

In some cases reward signal is assigned only at the end of experience. For example robot could be wondering around the maze receiving a reward only if the exit is found. These cases are usually harder to solve, but easier to analyze. We call this kind of situation an *episodic* problem.

Policies A policy is a rule specifying the behaviour of an agent. We identify a policy μ by a vector of parameters θ. *Policy class* Θ is some constrained set of policies parametrized by $\theta \in \Theta$. Sometimes, we call a policy with parameterization θ simply a "policy θ". Generally speaking, in a POMDP, a policy $\mu : H \times A \to [0, 1]$ is a rule specifying the probability of performing the action at each time step t as a function of the whole previous experience h^t, i.e., the complete sequence of observation-action pairs since time 0. This kind of policy is only of theoretical interest and is not feasible, since the number of possible experiences grows exponentially with time.

We will consider two simplifications. The first is a *reactive policy* (RP), sometimes called *state-free* or *memoryless*, which chooses an action based only on the last observation. Reactive policies can be *deterministic* or *stochastic*, mapping the last observation into an action or a probability distribution over actions, respectively. Formally, stochastic policy $\mu(a, o, \theta)$ specifies a probability of taking an action a, while deterministic reactive policy $\mu_d(s, \theta)$ actually specifies which action to take. The second is to use a memory, or an *internal state* of the controller to remember some crucial features of previous experience, or perhaps to simply remember a few previous observations and actions. Such a policy takes into account both the new observation and the internal state of the controller when choosing an action. Most of this work is concerned with stochastic policies. It is assumed that for any stochastic policy the probability of choosing any action a is bounded away from zero: $0 \leq \underline{c} \leq \Pr(a|h, \theta)$, for any $h \in H$ and $\theta \in \Theta$.

Value Any policy $\theta \in \Theta$ defines a conditional distribution $\Pr(h|\theta)$ on the set of all experiences H. The value of policy θ is the expected return according to the probability distribution induced by this policy on the space of experiences:

$$V(\theta) = E_\theta\left[R(h)\right] = \sum_{h \in H} \big(R(h)\Pr(h|\theta)\big), \tag{1.1}$$

where for brevity we introduce the notation E_θ for $E_{\Pr(h|\theta)}$. We assume that the policy value is bounded by V_{max}. That means of course that returns are also bounded by V_{max} since value is a weighted sum of returns. Another implicit assumption here is that there is some given starting state of the environment or alternatively a distribution over initial states which is factored into the expectation. Sometimes it makes sense to define a value of some policy θ at a given state s, denoted $V(\theta, s)$. This value corresponds to the return accumulated if the agent starts in state s and executes the policy θ.

Optimality It is the agent's objective to find a behavior which optimizes some long-run measure of feedback. Formally, it means to find a policy θ^* with optimal value: $\theta^* = \mathrm{argmax}_\theta V(\theta)$. It is a remarkable property of MDPs [133] that there exists an optimal deterministic reactive policy $\theta^* : S \rightarrow A$. Unfortunately, this kind of policy cannot be used in the partially observable framework, because of the uncertainty in the current state of the process. The optimal deterministic policy in POMDPs might have to be represented using an infinitely large internal state.

1.2 Solving Markov Decision Processes

Bellman equation It is important for the development of formalism to first consider the case of finding the optimal policy for an MDP in the case when the agent hypothetically knows the model of the environment, which means the knowledge of the reward and transition functions ρ and T. If the agent starts in some state s and executes the optimal policy θ^*, the infinite discounted sum of collected reward $V(s,\theta^*) = \max_\theta \mathrm{E}_\theta\left[\sum_{t=0}^{\infty}\gamma^t r(t)\right]$, which we call the optimal value of the state s. It can be shown [133] that this function is the unique solution to the *Bellman equation*:

$$V(s,\theta^*) = \max_{a\in A}\left(\rho(s,a) + \gamma\sum_{s'\in S} T(s,a,s')V(s',\theta^*)\right), \quad \forall s \in S. \quad (1.2)$$

The intuition behind this equation is that the optimal value of the state is the sum of the immediate reward for the optimal action in this state and the expected discounted value of the next step. Note that if the agent was given the optimal value function, it would retrieve the optimal policy $\mu_d(s,\theta^*)$ by:

$$\mu_d(s,\theta^*) = \arg\max_{a\in A}\left(\rho(s,a) + \gamma\sum_{s'\in S} T(s,a,s')V(s',\theta^*)\right), \forall s \in S. \quad (1.3)$$

However, the model of the environment is completely unknown to the agent in most problems considered in reinforcement learning. There are two general directions in finding the optimal policy without the model: learning a model and using it to derive an optimal policy (*model-based*); and learning an optimal policy without learning a model (*model-free*).

Q-learning *Q-learning* by Watkins [183, 184] is an example of a model-free algorithm, which is very popular in machine learning. Analogous to the optimal state value, we define the optimal state-action value $Q^*(s,a)$ as expected discounted reinforcement received by starting at state s and taking an optimal action a, then continuing according to the optimal policy $\mu_d(s,\theta^*)$. The optimal state-action value is a solution to the following equation, which is a restatement of the original Bellman equation:

$$Q^*(s,a) = \rho(s,a) + \gamma\sum_{s'\in S} T(s,a,s')\max_{a'\in A} Q^*(s',a'), \forall a \in A, s \in S. \quad (1.4)$$

Note that $V(s, \theta^*) = \max_{a \in A} Q^*(s, a)$ and therefore an optimal policy is given by $\mu_d(s, \theta^*) = \arg\max_{a \in A} Q^*(s, a)$, so we can compute $\mu_d(s, \theta^*)$ from Q^* without knowing ρ and T.

The basic idea of the algorithms is that we maintain the set of value estimates for state-action pairs, which represent the forecast of cumulative reinforcement the agent would get starting at a given state by performing a given action, and following a particular policy from there on. These values guide the behavior, providing an opportunity to do a feasible local optimization at every step.

Given an experience in the world, characterized by starting state s, action a, reward r, resulting state s' and next action a', the state-action pair update rule for Q-learning is

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right] . \tag{1.5}$$

While learning, any policy can be executed, as long as on an infinitely long run each action is guaranteed to be taken infinitely often at each state, and the learning rate α is decreased appropriately. Under those conditions the estimates $Q(s, a)$ will converge to the optimal values $Q^*(s, a)$ with probability one [175, 183].

SARSA The SARSA algorithm[3] differs from the classical Q-learning algorithm [183] in that, rather than using the maximum Q-value from the resulting state as an estimate of that state's value, it uses the Q-value of the resulting state and the action that was actually chosen in that state. Thus, the values learned are sensitive to the policy being executed.

Given an experience in the world, characterized by starting state s, action a, reward r, resulting state s' and next action a', the state-action pair update rule for SARSA(0) is

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right] . \tag{1.6}$$

In truly Markov domains, Q-learning is usually the algorithm of choice. In fact, Q-learning (QL) can be shown to fail to converge on very simple non-Markov domains [158]. Policy-sensitivity is often seen as a liability, because it makes issues of exploration more complicated.

[3] SARSA is an on-policy temporal-difference control learning algorithm [167].

However, in non-Markov domains, policy-sensitivity is actually an asset. Because observations do not uniquely correspond to underlying states, the value of a policy depends on the distribution of underlying states given a particular observation. But this distribution generally depends on the policy. So, the value of a state, given a policy, can only be evaluated *while executing that policy.* Note that when SARSA is used in a non-Markovian environment, the symbols s and s' in equation (1.6) represent sensations, which usually can correspond to several states.

The SARSA algorithm can be augmented by an eligibility trace, to yield the so-called SARSA(λ) algorithm (see Sutton and Barto [167] for details). SARSA(λ) describes a class of algorithms, where appropriate choice of λ is made depending on the problem. With the parameter λ set to 0, SARSA(λ) becomes regular SARSA. With λ set to 1, it becomes a pure Monte-Carlo method, in which, at the end of every trial, each state-action pair is adjusted toward the cumulative reward received on this trial after the state-action pair occurred. Pure Monte-Carlo algorithm makes no attempt to satisfy Bellman equation relating the values of subsequent states. Since it is often impossible to satisfy the Bellman equation in partially observable domains, Monte-Carlo is a reasonable choice. Generally, SARSA(λ) with a large value of λ seems to be the most appropriate among the conventional reinforcement-learning algorithms for solving partially observable problems.

Learning in POMDPs There are many approaches to learning to behave in partially observable domains. They fall roughly into three classes: optimal memoryless, finite memory, and model-based.

The first strategy is to search for the best possible memoryless policy. In many partially observable domains, memoryless policies can actually perform fairly well. Basic reinforcement-learning techniques, such as Q-learning [184], often perform poorly in partially observable domains, due to a very strong Markov assumption. Littman showed [86] that finding the optimal memoryless policy is NP-hard. However, Loch and Singh [89] effectively demonstrated that techniques, such as SARSA(λ), that are more oriented toward optimizing total reward, rather than Bellman residual, often perform very well. In addition, Jaakkola, Jordan, and Singh [67] have developed an algorithm for finding stochastic memoryless policies, which can perform significantly better than deterministic ones [157].

One class of finite memory methods are the finite-horizon memory methods, which can choose actions based on a finite window of previous observations. For many problems this can be quite effective [98, 136]. More generally, we may use a finite-size memory, which can possibly be infinite-horizon (the systems remembers only a finite number of events, but these events can be arbitrarily far in the past). Wiering and Schmidhuber [185] proposed such an approach, involving learning a policy that is a finite sequence of memoryless policies.

Another class of approaches assumes complete knowledge of the underlying process, modeled as a POMDP. Given a model, it is possible to attempt optimal solution [68], or to search for approximations in a variety of ways [58, 59, 60, 104]. The classical—Bayesian—approach is based on updating the distribution of probabilities for the environment to be in a particular state (or the so-called belief) at each time step, using the most recent observations [33, 60, 32, 68]. The problem is reformulated as a new MDP using belief-states instead of the original states. Unfortunately, the Bayesian calculation is highly intractable as it searches the continuous space of beliefs and considers multitude of possible sequence of observations. These methods can, in principle, be coupled with techniques, such as variations of the Baum-Welch algorithm [134, 135] for learning the model, to yield model-based reinforcement-learning systems.

To conclude, there are many efficient algorithms for the case when the agent has perfect information about the environment. An optimal policy is described by mapping the last observation into an action and can be computed in polynomial time in the size of the state and action spaces and the effective time horizon [18]. However in many cases the environment state is described by a vector of several variables, which makes the environment state size exponential in the number of variables. Also, under more realistic assumptions, when a model of environment dynamics is unknown and the environment's state is not observable, many problems arise. The optimal policy could potentially depend on the whole history of interactions and for the undiscounted finite horizon case finding it is PSPACE-complete [119].

1.3 Optimization in Policy Space

Policy search is a family of methods which perform direct search in policy space. Powell [129] describes many direct search methods. The general philosophy behind these methods is reminiscent of Vapnik's motto for support vector machines [179]: "Do not try to solve a complex problem as a step towards solving a simpler one". For reinforcement learning, it essentially means abandoning the hope of estimating the model of the environment or Markov chain's transition matrix. Nether do we intend to estimate state or state-action values with certain semantics as defined earlier in section 1.2. Instead the encoding of policy is chosen, which constrains a search space and optimal or near-optimal elements of this space are sought.

Policy is understood in a very broad, "black-box" sense – as far as it defines how to choose actions given a complete history, it is a policy. In this section we will review several ways of policy encoding and corresponding policy search methods guided by some bias in search, even though these methods might have not been necessarily classified as policy search methods by their authors. Policy search methods could for example search the space of programs in some language [143, 147, 146, 14, 15], space of encodings of the structure of neural networks [107, 108, 109, 127] or simply parameters in a look-up table or stochastic finite state controller [68, 123, 104].

Constraint on the search space could constitute just specification of a policy encoding language, or specification of a particular range of parameters, or a fixed connection in some part of a neural network. Orthogonal to the way policy is represented, the approach for searching the space of policies could also be chosen from a wide variety of optimization methods. An application of evolutionary algorithms to the reinforcement learning problem, emphasizing alternative policy representations, credit assignment methods, and problem-specific genetic operators [109].

Baum argues that the mind should be viewed as an economy of idiots [14, 15]. His "Hayek Machine" searches the space of programs encoding behavior. Working with such scalable problems as a simulated block world, this method begins by solving simple instances. Solutions to these simpler problem instances are used to form a bias, which in turn facilitates a search for solution of an larger instance of the same problem. The search is done in a way reminiscent of evolutionary

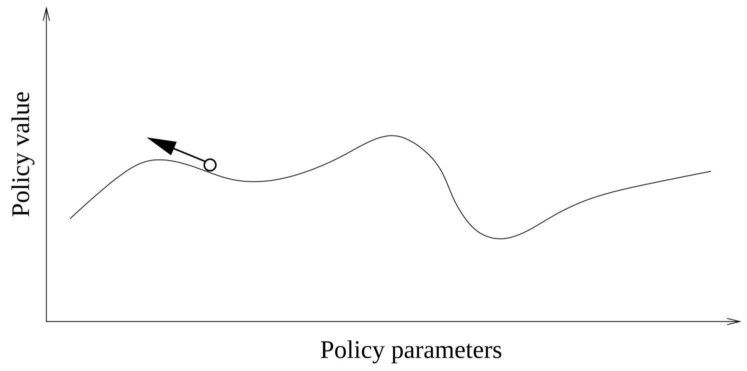


Fig. 1.3: Learning as optimization problem.

programming and genetic algorithms [39] while the notion of fitness is given a monetary interpretation and "conservation of money" along with other economics analogies are employed.

The goal of RL is essentially finding policy parameters that maximize a noisy objective function. The Pegasus method converts this stochastic optimization problem into a deterministic one, by using fixed start states and fixed random number sequences for comparing policies [111]. Strens and Moore[164] developed this method using paird tests. By adapting the number of trials used for each policy comparison they accelerate learning.

One way to search the policy space is to try out all possibilities. Traversing the space has to have a particular order, or bias, reflecting some knowledge or assumptions regarding solutions in the search space. E.g. one might assume that low complexity solutions are more likely, according to some measure of the policy's complexity. Using a complexity measure derived from both Kolmogorov complexity and computational complexity is the basis of *Levin search* [147].

Littman [87] introduced branch-and-bound methods to RL. Branch-and-bound acts by constructing partial solutions and using these to exclude parts of the search space from consideration as containing only inferior options. A partial solution could be a policy specified for only a part of the state space. It is sometimes possible to establish a minimal performance for such policy independent of the remaining part of it. If such minimal performance level is higher that an upper performance level for some other partial policy, this second branch could be excluded from further consideration. A set of branch-and-bound heuristics is used to decide which branches of search process to pursue and how to establish upper bounds which allow exclusion of bigger parts of a search space quicker.

In this dissertation we focus on policy search by following the gradient. The general idea behind policy search by gradient is to start with some policy, evaluate it and make an adjustment in the direction of the gradient of a policy's value function as illustrated by figure 1.3. Very different optimization methods can be applied such as genetic algorithms or simulated annealing. In the following section we review related work in the field of gradient methods. Detail of the method of gradient ascent for policy search is left for the next chapter.

1.4 Related Gradient Methods

The gradient approach to reinforcement learning was introduced by Williams [187, 188, 189] in the REINFORCE algorithm, which estimates the gradient of expected cumulative reward in *episodic* tasks. For such tasks there is an identified target state of the environment, such that, upon encountering that state, the algorithm returns an estimate of the gradient based on the trace of the reinforcement signal accumulated during the episode.

Williams' REINFORCE was generalized to optimize the average reward criterion in MDPs based on a single sample path by Marbach and Tsitsiklis [95, 92]. Parameters can be updated either during visits to a certain recurrent state or on every time step. Their algorithm has the established convergence of the performance metric with probability one. Baird and Moore [8, 9] presented an algorithm called VAPS, which combines value-function with policy gradient approaches. VAPS relies on the existence of a recurrent state to guarantee convergence. Algorithms considered in the rest of this work were motivated by VAPS. Therefore it receives a special attention elsewhere in this document.

Recently there was parallel work by Tsitsiklis and Konda [79] and Sutton, *et al.* [168] which analyzes gradient ascent algorithms in the framework of "Actor-Critic" methods (introduced by Barto [13]). Due to the "critic" part, which makes sense mostly under conditions of complete observability, that work applies to MDPs with function approximation. They show that since the number of parameters in the representation of a policy ("actor") is often small compared to the number of states, the value-function ("critic") must not be represented in a high-dimensional space, but rather in a lower dimensional space spanned by the basis, which is determined by the representation of the policy. Using this result they analyze and prove convergence to the locally optimal policy.

Bartlett and Baxter [16] introduce their version of estimating the gradient, called GPOMDP. They also make an assumption that every state is recurrent, and estimate the gradient in one (infinitely long) trial. The specifics of their approach is in using a conjugate gradient algorithm which utilizes estimates made by GPOMDP and a line search instead of decreasing learning rate. The local convergence of GPOMDP with probability one is proven. For the sake of theoretical guarantees, both methods mentioned use an average (undiscounted) reward criterion, whose gradient is estimated through the estimate of the state occupancy distribution. One of the issues with algorithms estimating the gradient is that estimates tend to have high variance resulting in a slow convergence. Tsitsiklis and Marbach [96] propose the solution with discounting, which introduces bias into the estimate, and establish bounds for the convergence properties of this method.

Grudic and Ungar [57, 56] extend methods by Tsitsiklis and Konda [79] and Sutton, *et al.* [168]. They provide an interesting example of leveraging the knowledge of the domain by choosing an appropriate low-dimensional policy representation. They present a *Boundary Localized Reinforcement Learning* (BLRL) method which is very effective for cases of completely observable MDPs with continuous state spaces and a few actions, such that in the optimal policy large regions of the state space correspond to the same action. In the *Action Transition Policy Gradient* they are estimating *relative* values of actions and show that performance could be orders of magnitude better than that of algorithms employing a more naive representation. This idea can be also viewed in light of learning feature extraction.

Ascending along the direction of the gradient requires choosing a step size. Amari [2] revolutionized the field of optimization by gradient methods, introducing geometric considerations for calculating the gradient. Sham Kakade [70] developed a pioneering approach of applying natural gradient methods to policy search in reinforcement learning. He gives empirical evidence of learning acceleration when the "natural" policy gradient is contrasted with the regular policy gradient.

Chapter 2

Policies with Memory

Summary In order for an agent to perform well in partially observable domains, it is usually necessary for actions to depend on the history of observations. In this chapter we present a controller based on the GAPS algorithm combined with memory. We first explore a *stigmergic* approach, in which the agent's actions include the ability to set and clear bits in an external memory, and the external memory is included as part of the input to the agent. We also consider the development of the GAPS algorithm for the case of finite state controllers, in which memory is a part of the controller. We illustrate advantages of both architectures as compared to other existing approaches by empirical results for several domains.

2.1 Gradient Ascent for Policy Search

Here we consider the case of a single agent interacting with a POMDP. For now, we will not make any further commitment to details of the policy's architecture, as long as it defines the probability $\Pr(a, h^t, \theta)$ of action a given past experience h^t as a continuous differentiable function of some vector of parameters θ. Later, we will consider various architectures of the controller, which factor additional information into the expression for this probability; e.g., the agent's policy μ could depend on some finite internal state.

The objective of an agent is to choose a strategy that maximizes the expected cumulative reward. We will assume a single starting state, since for the case of the distribution over initial state, one can introduce an extra starting state and have an initial transition from this state according to this distribution. Remember that value of a strategy θ (see equation 1.1) is

$$V(\theta) = E_\theta\left[R(h)\right] = \sum_{h \in H} \big(R(h)\Pr(h|\theta)\big) .$$

Note that the policy is parametrized by a vector $\theta = \{\theta_1, \ldots, \theta_n\}$. If we could calculate the derivative of $V(\theta)$ for each θ_k, it would be possible to do exact gradient ascent (see a paper by Meuleau *et al.* [104]) on value $V()$ by making updates

$$\Delta\theta_k = \alpha \frac{\partial}{\partial\theta_k} V(\theta) .$$

Since $R(h)$ does not depend on θ,

$$\frac{\partial}{\partial\theta_k} V(\theta) = \sum_{h \in H} \left(R(h) \frac{\partial}{\partial\theta_k} \Pr(h|\theta)\right) .$$

Let us examine the expression for $\Pr(h|\theta)$. The Markov assumption in POMDPs warrants that $\Pr\big(h|\theta\big)$

$$
\begin{aligned}
&= \Pr\big(s(0)\big) \prod_{t=1}^{T} \Pr\big(o(t)|s(t)\big) \Pr\big(a(t)|o(t),\theta\big) \Pr\big(s(t+1)|s(t),a(t)\big) \\
&= \left[\Pr\big(s(0)\big) \prod_{t=1}^{T} \Pr\big(o(t)\big|s(t)\big) \Pr\big(s(t+1)\big|s(t),a(t)\big)\right] \left[\prod_{t=1}^{T} \Pr\big(a(t)\big|o(t),\theta\big)\right] \\
&= \Phi\big(h\big)\Psi\big(\theta,h\big) .
\end{aligned}
$$

$\Phi(h)$ is the factor in the probability related to the part of the experience dependent on the environment, that is unknown to the agent and can only be sampled. $\Psi(\theta, h)$ is the factor in the probability related to the part of the experience, dependent on the agent, that is known to the agent and can be computed (and differentiated). Note that $\Pr\big(h|\theta\big)$ can be broken up this way both when controller executes a reactive policy and a policy with internal state (see for example the derivation for internal state sequences in Shelton's dissertation [151]). Taking into account this decomposition, we get:

$$\frac{\partial}{\partial\theta_k} V(\theta) = \sum_{h \in H} \left(R(h)\Phi(h) \frac{\partial}{\partial\theta_k} \Psi(\theta,h)\right) .$$

Rewriting $\Psi(\theta, h)$ as a product of policy terms $\mu(a(t), o(t), \theta)$ and taking into account that in a course of some experience h, the same observation o and same observation-action pair (o, a) could be encountered several times, reflected by counter variables N^h_o and N^h_{oa} respectively, we have

$$\Psi(\theta, h) = \prod_{t=1}^{T} \Pr\left(a(t) \middle| o(t), \theta\right) = \prod_{o \in O, a \in A} \mu(a, o, \theta)^{N^h_{oa}} .$$

Let us consider the case when for some parameter θ_k and $j \neq k$ we have $\frac{\partial}{\partial \theta_j} \mu(a, o, \theta) = 0$, e.g. when there is one parameter θ_{oa} for each observation-action pair (o, a). For this case $\frac{\partial}{\partial \theta_{oa}} \Psi(\theta, h)$

$$\begin{aligned}
&= \left(\frac{\partial \mu(a, o, \theta)}{\partial \theta_{oa}} \right) N^h_{oa} \mu(a, o, \theta)^{N^h_{oa} - 1} \times \prod_{o' \neq o, a' \neq a} \mu(a', o', \theta)^{N^h_{o'a'}} \\
&= \left(\frac{\partial \mu(a, o, \theta)}{\partial \theta_{oa}} \right) N^h_{oa} \frac{\mu(a, o, \theta)^{N^h_{oa}}}{\mu(a, o, \theta)} \times \prod_{o' \neq o, a' \neq a} \mu(a', o', \theta)^{N^h_{o'a'}} \\
&= N^h_{oa} \left(\frac{\partial \mu(a, o, \theta)}{\partial \theta_{oa}} \right) \frac{1}{\mu(a, o, \theta)} \times \prod_{o' \in O, a' \in A} \mu(a', o', \theta)^{N^h_{o'a'}} \\
&= N^h_{oa} \left(\frac{\partial \mu(a, o, \theta)}{\partial \theta_{oa}} \right) \frac{1}{\mu(a, o, \theta)} \times \Psi(\theta, h) \\
&= N^h_{oa} \left(\frac{\partial \ln \mu(a, o, \theta)}{\partial \theta_{oa}} \right) \times \Psi(\theta, h) .
\end{aligned}$$

Note the presence of $\mu()$ in the denominator, which requires an assumption of non-zero probability of any action under any encoding θ. We can finally express the derivative as

$$\frac{\partial}{\partial \theta_{oa}} V(\theta) = \sum_{h \in H} R(h) \left(\Pr(h|\theta) N^h_{oa} \frac{\partial}{\partial \theta_{oa}} \ln \mu(a, o, \theta) \right) .$$

However in the spirit of reinforcement learning, we cannot assume the knowledge of a world model that would allow us to calculate $\Psi(h)$ and subsequently $\Pr(h \mid \theta)$, neither are we able to perform summation over all possible experiences $h \in H$. Henceforth we must retreat to *stochastic gradient ascent* instead.

Remark **2.1.1** *We have worked with the return function* $R(h)$*, not taking into account specifics of how rewards are assembled into* *return. For the case of episodic tasks (see p. 11), there is no improvement possible. In some other cases, a reward is received at intermediate steps, so an algorithm with better credit assignment can be obtained (also see technical report [103]). For the case of infinite horizon discounted criterion with discount factor* $\gamma \in [0, 1)$ *and a strategy* θ, *the value (see equation 1.1) is* $V(\theta) = \sum_{t=1}^{\infty} \gamma^t E_\theta [r(t)]$. *Following the derivation by Baird and Moore [9] this value can be rewritten as* $V(\theta) = \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H_t} \Pr(h \mid \theta) r(t, h)$. *Let us examine the derivative* $\frac{\partial V(\theta)}{\partial \theta_k}$

$$\begin{aligned} =&\sum_{t=1}^{\infty} \gamma^t \sum_{h \in H_t} \left[r(t,h) \frac{\partial}{\partial \theta_k} \Pr(h \mid \theta) \right] \\ =&\sum_{t=1}^{\infty} \gamma^t \left[\sum_{h \in H_t} \Pr(h \mid \theta) r(t,h) \times \sum_{\tau=1}^{t} \frac{\partial \ln \Pr\left(a(\tau,h) | h^{\tau-1}, \theta\right)}{\partial \theta_k} \right]. \end{aligned} \tag{2.1}$$

We sample from the distribution of histories by interacting with the environment, and calculate during each trial an estimate of the gradient, for all t *accumulating the quantities:*

$$\gamma^t r(t,h) \sum_{\tau=1}^{t} \frac{\partial \ln \Pr\left(a(\tau,h) \mid h^{\tau-1}, \theta\right)}{\partial \theta_k}. \tag{2.2}$$

For a particular policy architecture, this can be readily translated into a gradient ascent algorithm that is guaranteed to converge to a local optimum of $V()$. We refer to this algorithm as GAPS, for *gradient ascent policy search*. The rest of the work presented in this thesis mainly concerns the application of gradient ascent algorithms to various architectures of the controller. In particular we will explore the performance of controllers with external memory, the stochastic finite state machine and distributed controllers of cooperating agents. We will present the resulting formulae as we describe architectures in further chapters.

2.2 GAPS with a Lookup Table

In this section, we present GAPS, for the case in which policy parameters are stored in a *look-up table*. That is, there is one parameter θ_{oa} for each observation-action pair (o, a). Note that it is not necessary to use the sequence-based gradient in a look-up table implementation of

QL or SARSA, as long as it is confined to a Markovian environment. However, it makes sense to use it in the context of POMDPs. Under this hypothesis, the exploration trace associated with each parameter θ_{oa} will be written $E(o, a, t)$.

We will also focus on a very popular rule for randomly selecting actions as a function of their θ-parameter, namely the *Boltzmann law* (also known as *soft-max*), where ζ is a temperature parameter:[1]

$$\mu(a, o, \theta) = \Pr\left(a(t) = a \middle| o(t) = o, \theta\right) = \frac{\exp\left(\theta_{oa}/\zeta\right)}{\sum_{a'} \exp\left(\theta_{oa'}/\zeta\right)} > 0. \quad (2.3)$$

Under this policy encoding we get:

$$\frac{\partial \ln \mu(a, o, \theta)}{\partial \theta_{o'a'}} = \begin{cases} 0 & \text{if } o' \neq o, \\ -\frac{1}{\zeta}\mu\left(a', o, \theta\right) & \text{if } o' = o \text{ and } a' \neq a, \\ \frac{1}{\zeta}\left[1 - \mu(a, o, \theta)\right] & \text{if } o' = o \text{ and } a' = a. \end{cases}$$

Let us also assume that the problem is an *episodic* task, i.e., the reward is always zero except when we reach an absorbing goal state. Then the exploration trace $E(o, a, t)$ takes a very simple form:

$$E(o, a, t) = \frac{\left[N^h_{oa}(t) - N^h_o(t)\mu\left(a, o, \theta\right)\right]}{\zeta} = \frac{\left[N^h_{oa}(t) - E[N^h_{oa}(t)]\right]}{\zeta}, \quad (2.4)$$

where $N^h_{oa}(t)$ is the number of times that action a has been executed given an observation o up until time t, $N^h_o(t)$ is the number of times that observation o has been obtained up until time t, and $E[N^h_{oa}(t)]$ represents the expected number of times one should have performed action a given an observation o, knowing the exploration policy and previous experience.

As a result of equation (2.4), GAPS using look-up tables and Boltzmann exploration reduces to a very simple algorithm, presented in table 2.1. At each time-step where the current trial does not complete, we just increment the counter N_{oa} of the current observation-action pair. When the trial completes, this trace is used to update all the parameters, as described above.

Remark **2.2.1** *It is interesting to try to understand the properties and implications of this simple rule. First, a direct consequence*

[1]Note that Baird and Moore [9] use an unusual version of the Boltzmann law, with $1 + e^x$ in place of e^x in both the numerator and the denominator. We have found that it complicates the mathematics and worsens the performance, so we will use the standard Boltzmann law throughout.

Tab. 2.1: The GAPS algorithm for RP with a look-up table.

```
Initialize policy: For all $(o, a)$: $\theta_{oa} \leftarrow 0$
For $i = 1$ to $n$: (make $n$ trials)
    • Beginning of the trial:
        $R \leftarrow 0$
        for all $(o, a)$:
            $N_o \leftarrow 0$, $N_{oa} \leftarrow 0$
    • At each time step $t$ of the trial:
        Get observation $o(t)$
        inc($N_o$)
        Draw next $a(t)$ from $\mu(a(t), o(t), \theta)$
        inc($N_{oa}$)
        Execute $a(t)$, get reward $r(t)$
        $R \leftarrow R + \gamma^t r(t)$
    • End of the trial-update:
        for all $(o, a)$:
            $\theta_{oa} \leftarrow \theta_{oa} + \alpha R\,(N_{oa} - \mu(a, o, \theta) N_o)$
```

is that when something surprising happens, the algorithm adjusts the unlikely actions more than the likely ones. In other words, this simple procedure is very intuitive, since it assigns credit to observation-action pairs proportional to the deviation from the expected behavior, which we call "surprise"*. In principle, this is similar to the way dopamine bonuses are generated in the brain as suggested by Schultz (see [150, 149]). Note that* SARSA(λ) *is not capable of such a discrimination. This difference in behavior is illustrated in the simulation results.*

A second interesting property is that the parameter updates for observation-action pair tend to 0 as the length of the trial tends to infinity. This also makes sense, since the longer the trial, the less the final information received (the final reward) is relevant in evaluating each particular action. Alternatively, we could say that when too many actions have been performed, there is no reason to attribute the final result more to one of them than to others. Finally, unlike with Baird and Moore's version of the Boltzmann law, the sum of the updates to the parameters on every step is zero. This makes it more likely that the parameters will stay bounded.

2.3 Controller with External Memory

Stigmergy In this work, we pursue an approach based on *stigmergy.* The term is defined in the Oxford English Dictionary [155] as "The process by which the results of an insect's activity act as a stimulus to further activity," and is used in the mobile robotics literature [17] to describe activity in which an agent's changes to the world affect its future behavior, usually in a useful way. One form of stigmergy is the use of external memory devices. We are all familiar with practices such as making grocery lists, tying a string around a finger, or putting a book by the door at home so you will remember to take it to work. In each case, an agent needs to remember something about the past and does so by modifying its external perceptions in such a way that a memoryless policy will perform well.

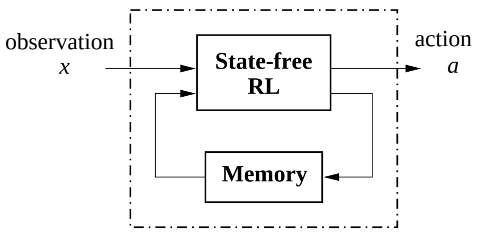


Fig. 2.1: The stigmergic architecture.

We can apply this approach to the general problem of learning to behave in partially observable environments. Figure 2.1 shows the architectural idea. We think of the agent as having two components: one is a set of memory bits; the other is a reinforcement-learning agent. The reinforcement-learning agent has as input the observation that comes from the environment, augmented by the memory bits. Its output consists of the original actions in the environment, augmented by actions that change the state of the memory. If there are sufficient memory bits, then the optimal memoryless policy for the internal agent will cause the entire agent to behave optimally in its partially observable domain.

There are two alternatives for designing an architecture with external memory. The first is to *augment* the action space with actions that change the content of one of the memory bits (adds L new actions if there are L memory bits); changing the state of the memory may require multiple steps. The second is to *compose* the action space with the set of all possible values for the memory (the size of the action space is then multiplied by 2^L, if there are L bits of memory). In this case, changing the external memory is an instantaneous action that can be done at each time step in parallel with a primitive action, and hence we can reproduce the optimal policy of some domains without taking additional memory-manipulation steps. Complexity considerations usually lead us to take the first option. It introduces a bias, since we have to lose at least one time-step each time we want to change the content of the memory. However, it can be fixed in most algorithms by not discounting memory-setting actions.

The external-memory architecture has been pursued in the context of classifier systems [23] and in the context of reinforcement learning by Littman [86] and by Martín [97]. Littman's work was model-based; it assumed that the model was completely known and did a branch-and-bound search in policy space. Martín worked in the model-free reinforcement-learning domain; his algorithms were very successful at finding good policies for very complex domains, including some simulated visual search and block-stacking tasks. However, he made a number of strong assumptions and restrictions: task domains are strictly goal-oriented; it is assumed that there is a deterministic policy that achieves the goal within some specified number of steps from every initial state; and there is no desire for optimality in path length.

The success of Martín's algorithm on a set of difficult problems is inspiring. However it has restrictions and a number of details of the algorithm seem relatively *ad hoc*. At the same time, Baird and Moore's work on VAPS [9], a general method for gradient descent in reinforcement learning, appealed to us on theoretical grounds. This work is the result of attempting to apply VAPS algorithms to stigmergic policies, and understanding how it relates to Martín's algorithm. In this process, we have derived a much simpler version of VAPS for the case of highly non-Markovian domains: we calculate the same gradient as VAPS, but with much less computational effort. We call the new algorithm GAPS.

2.4 Experiments with Stigmergic Controllers

Load-unload Consider the load-unload problem represented in Figure 2.2. In this problem, the agent is a cart that must drive from an *Unload* location to a *Load* location, and then back to *Unload*. This problem is a simple POMDP with a one-bit hidden variable that makes it partially observable (the agent cannot see whether it is loaded or not; aliased states, which have the same observation, are grouped by dashed boxes on figure 2.2). It can be solved using a one-bit external memory: we set the bit when we make the *Unload* observation, and we go right as long as it is set to this value and we do not make the *Load* observation. When we do make the *Load* observation, we clear the bit and we go left as long as it stays cleared, until we reach state 9, getting a reward.

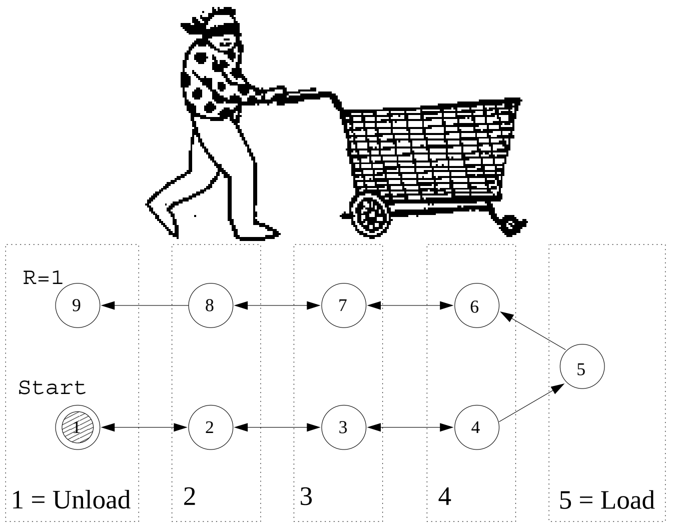


Fig. 2.2: The state-transition diagram of the load-unload problem.

We have experimented with SARSA and GAPS on several simple domains. Two are illustrative problems previously used in the reinforcement-learning literature: Baird and Moore's problem [9], created to illustrate the behaviour of VAPS, and McCallum's 11-state maze [98], which has only six observations. The third domain is an instance of a five-location load-unload problem (fig. 2.2). The last domain is a variant of load-unload designed specifically to demonstrate a situation in which GAPS might outperform SARSA. In this variant of the load-unload problem a second loading location has been added, and the agent is punished instead of rewarded if it gets loaded at the wrong location. The state space is shown in figure 2.3; states contained in a dashed box are observationally indistinguishable to the agent. The idea here is that there is a single action that, if chosen, ruins the agent's long-term prospects. If this action is chosen due to exploration, then SARSA(λ) will punish all of the action choices along the chain but GAPS will punish only that action.

Algorithmic Details and Experimental Protocol

All domains have a single starting state, except McCallum's problem, where the starting state is chosen uniformly at random. For each problem, we ran two algorithms: GAPS and SARSA(1). The optimal policy for Baird's problem is memoryless, so the algorithms were applied directly in that case. For the other problems, we augmented the input space with an additional memory bit, and added two actions: one for setting the bit and one for clearing it. We employed GAPS using look-up tables and the Boltzmann rule.

The learning rate α was determined by a parameter, α_0; the actual learning rate has an added factor that decays to zero over time: $\alpha = \alpha_0 + \frac{1}{10N}$, where N is trial number. The temperature was also decayed in an *ad hoc* way, from $\zeta_{\max}$ down to $\zeta_{\min}$ with an increment of $\Delta\zeta = \left(\frac{\zeta_{\min}}{\zeta_{\max}}\right)^{1/(N-1)}$ on each trial. In order to guarantee convergence of SARSA in MDPs, it is necessary to decay the temperature in a way that is dependent on the θ-parameters themselves [159]; in the POMDP setting it is much less clear what the correct decay strategy is. We have found that performance of the algorithm is not particularly sensitive to the temperature decay schedule.

Each learning algorithm was executed for 50 runs; each run consisted of N trials, which began at the start state and executed until a terminal state was reached or M steps were taken. The run terminated after M steps was given a reward of -1; M was chosen, in each case, to be 4 times the length of the optimal solution. At the beginning of each run, the parameters were randomly reinitialized to small values.

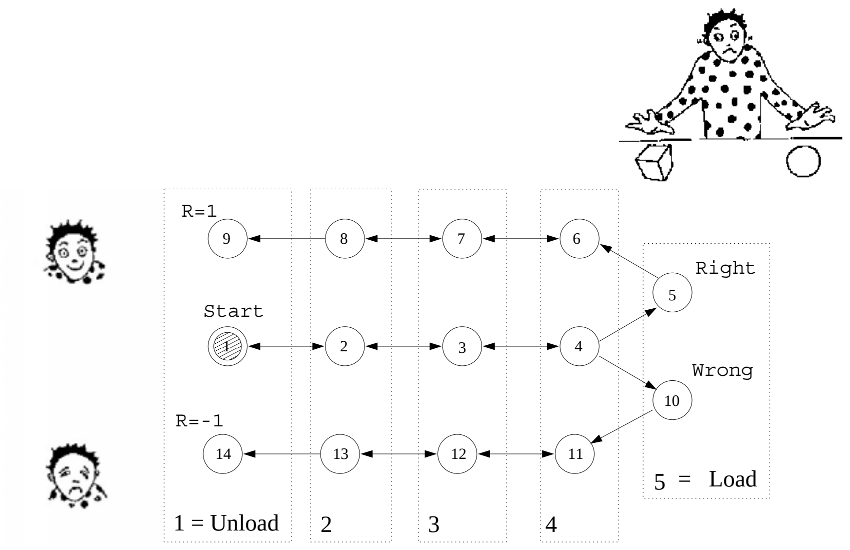
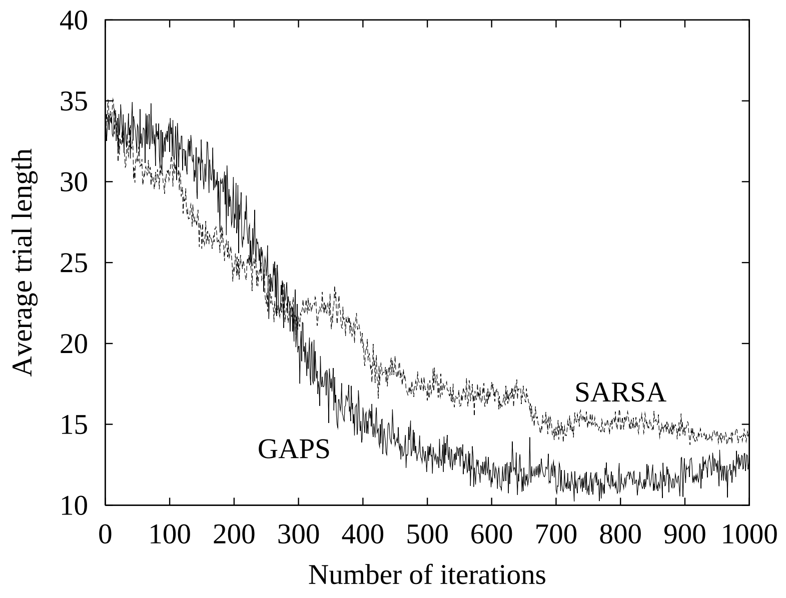


Fig. 2.3: The state-transition diagram of the problem with two loading locations.

Experimental Results

It was easy to make both algorithms work well on Baird's and McCallum's domains. The algorithms typically converged in fewer than a hundred trials to an optimal policy. Curiously enough we found that GAPS, which uses the true Boltzmann exploration distribution rather than the one described by Baird and Moore for VAPS, seems to perform significantly better, according to results presented in their paper [9].

Things were somewhat more complex with the last two problems (5 location load-unload with one or two loading locations). We experimented with parameters over a broad range and determined the following: Original VAPS requires a value of β equal or very nearly equal to 1; these problems are highly non-Markovian, so the Bellman error is not at all useful as a criterion; for similar reasons, $\lambda = 1$ is best for SARSA(λ); empirically, $\zeta_{max} = 1.0$ and $\zeta_{min} = 0.2$ worked well for GAPS in both problems, and $\zeta_{max} = 0.2$, $\zeta_{min} = 0.1$ worked well for SARSA(λ); a base learning rate of $\alpha_0 = 0.5$ worked well for both algorithms in both domains. Figure 2.4 shows learning curves for both algorithms, averaged over 50 runs, on the load-unload problem with one or two loading locations. Each run consisted of 1,000 trials. The



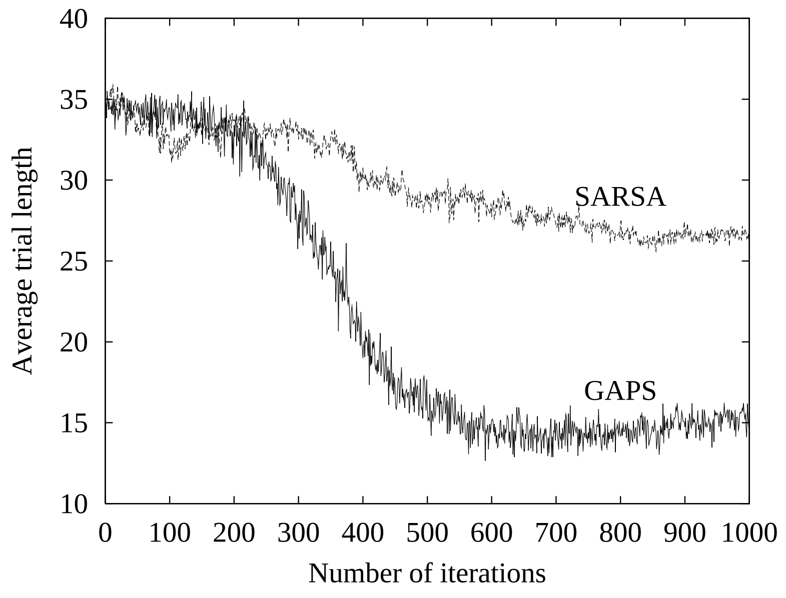


Fig. 2.4: Learning curves for a regular (top) and a modified (bottom) "load-unload".

vertical axis shows the number of steps required to reach the goal, with the terminated trials considered to have taken M steps.

On the original load-unload problem, the algorithms perform essentially equivalently. Most runs of the algorithm converge to the optimal trial length of 9 and stay there; occasionally, however it reaches 9 and then diverges. This can probably be avoided by decreasing the learning rate more steeply. When we add the second loading location, however, there is a significant difference. GAPS consistently converges to a near-optimal policy, but SARSA(1) does not. The idea is that sometimes, even when the policy is pretty good, the agent is going to pick up the wrong load due to exploration and get punished for it. SARSA will punish all the observation-action pairs equally; GAPS will punish the bad observation-action pair more due to the different principle of credit assignment.

As Martín and Littman showed, small POMDPs can be solved effectively using stigmergic policies. Learning reactive policies in highly non-Markovian domains is not yet well-understood. We have demonstrated that the GAPS algorithm can solve a collection of small POMDPs, and that although SARSA(λ) performs well on some POMDPs, it is possible to construct cases on which it fails. One could expect that the approach of augmenting controller with memory does not scale well, which is addressed further in section 2.5 for the application of the GAPS algorithm to the problem of learning general finite-state controllers (which encompass external-memory policies) for POMDPs.

2.5 Finite State Controllers

We limit our consideration in this chapter to cases in which the agent's actions may depend only on the current observation, or in which the agent has a finite internal memory. In general, to optimally control POMDPs one needs policies with memory. It could be necessary to have an infinite memory to remember all of the history [161, 160, 162]. Alternatively the essential information from the entire history of POMDPs can be summarized in a belief state [32] (a distribution over environment states), which is updated using the current observation and constitutes a *sufficient statistic* for the history [170]. In general this requires memory to store the belief state with an arbitrary precision. It can be shown [161, 160, 162] that in some cases, the belief state leads to representation of the optimal policy by a *policy graph* or *finite state controller*, where representing a policy requires finite memory.

There are many possibilities for constructing policies with memory. We have considered external memory in section 2.3 (also in our papers [123, 104]). In principle a policy could be represented by a neural network. However, feed-forward neural networks would not suffice since the optimal controller needs to be capable of retaining some information from the past history in its memory; i.e., to have internal state. This could be only achieved by a recurrent neural network. Such a recurrent network would have to output a probability distribution over actions and could be considered as a different encoding of a stochastic finite state controller. Choosing between recurrent neural networks and FSCs becomes a matter of aesthetics of representation. An extra argument for FSCs is that it is easier to analyze a policy or to construct an objective (sub)-optimal policy since FSCs have a clearer semantic explanation.

A finite state controller (FSC) for an agent with action space A and observation space O is a tuple $\langle M, \mu_a, \mu_m \rangle$, where M is a finite set of internal controller states, $\mu_m : M \times O \rightarrow \mathcal{P}(M)$ is the internal state transition function that maps an internal state and observation into a probability distribution over internal states, and $\mu_a : M \rightarrow \mathcal{P}(A)$ is the action function that maps an internal state into a probability distribution over actions. Note that we assume both the internal state transition function and the action function are stochastic and their derivatives exist and are bounded away from zero. Figure 2.5 depicts an influence diagram for an agent controlled by FSCs. Compare it to figure 1.2 where we did not have any definition of a policy. A FSC with a single state would constitute a degenerate case, corresponding to a memoryless policy (RP), or a direct link from $o(t)$ to $a(t)$.

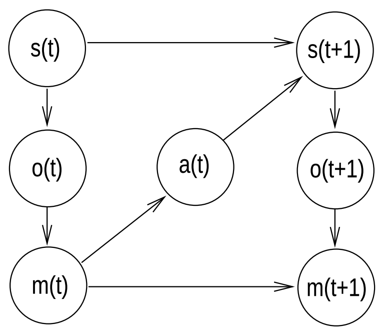


Fig. 2.5: An influence diagram for agent with FSCs in POMDP.

Tab. 2.2: The sequence of events in agent-environment interaction.

```
Initialize environment state $s(0)$, controller state $m(0)$
At each time step $t$ of the trial:
  • Generate observation $o(t)$ from $s(t)$ according to POMDP
    observation function $B(.)$
  • Draw new controller state $m(t)$ based on the current observation
    $o(t)$ and controller state $m(t-1)$ from $\mu_m(m(t), o(t), m(t-1), \theta)$
  • Draw next action $a(t)$ based on the current controller state
    $m(t)$ from $\mu_a(a(t), m(t), \theta)$
  • Execute action $a(t)$ and get reward $r(t)$
```

Remark **2.5.1** *We avoid specifying the initial state or the distribution over initial states since it is always possible to augment the given set of states with one extra state which is the starting state and from which a transition happens to one of the primary states. We also avoid defining the set of terminal or final states since the end of a control loop usually depends on a particular domain.*

Remark **2.5.2** *It is possible to define FSCs so that the action function depends on both the internal state and the current observation, but that alternative formulation would be expressively equivalent. An example of an expressively weaker formulation would be a action function which depends only on the current internal state, which in turn depends only on the current observation. This policy would be a kind of reactive policy where observations are first classified into a few categories.*

In partially observable environments, agents controlled by FSCs might not have enough memory to even represent an optimal policy which could, in general, require infinite state [162]. In this work, we concentrate on the problem of finding the (locally) optimal controller from the class of FSCs with some fixed size of memory. Table 2.2 presents the sequence of events in agent-environment interaction. In comparison to the definition of experience h (see section 1.1 on page 10) there is a new event occurring at each time step t: the controller enters some state $m(t)$. Let us assemble these events into a sequence $h_m = \langle m(0), \dots, m(t) \rangle$. There are two ways to handle this extra information regarding the agent's experience: disregard it all together or use it in the derivation of the gradient of expected return. It is important to see that $\Pr(a|h,\theta) \neq \Pr(a|h,h_m,\theta)$ since the former probability includes all possible ways in which controller could have chosen to perform action a via various internal states m: $\mu'(a,o,\theta) = \sum_m \Pr(a,m|h,h_m,\theta) = \sum_m \Pr(a|h,\theta) \times \Pr(m|h_m,\theta)$, while the latter corresponds to the one particular internal state: $\mu(a,o,\theta) = \mu_a(a,m,\theta) \times \mu_m(m,o,m',\theta)$. Learning the controller's parameters θ in the former case is reminiscent of learning a hidden Markov model (HMM) [135, 134] and is covered in the dissertation by Christian Shelton [151]. Here we consider the latter case when the agent keeps a record of the internal state.

2.6 GAPS with finite state controllers

In this section we derive the algorithm for learning FSCs via gradient ascent policy search. If we use look-up tables to store the parameters of the FSCs, then there is one parameter, denoted θ_{ma} corresponding to each possible internal state-action pair (m,a) and one parameter $\theta_{mom'}$ corresponding to each possible internal state transition given each observation (m,o,m'). Using Boltzmann law with a temperature parameter ζ (compare to equation (2.3)) we obtain:

$$\mu_a(a,m,\theta) = \Pr\big(a(t) = a \big| m(t) = m, \theta\big) = \frac{\exp(\theta_{ma}/\zeta)}{\sum_{a'} \exp(\theta_{ma'}/\zeta)} > 0 \,,$$

$$\begin{aligned}\mu_m(m,o,m',\theta) &= \Pr\big(m(t+1) = m' \big| m(t) = m, a(t) = a, \theta\big) \\ &= \frac{\exp(\theta_{mom'}/\zeta)}{\sum_{m''} \exp(\theta_{mom''}/\zeta)} > 0 \,.\end{aligned}$$

Since the encoding of action and internal state transition functions is independent, partial derivatives factor into separate terms:

$$\begin{aligned}\frac{\partial\mu(a,o,\theta)}{\partial\theta_{ma}} &= \frac{\partial\mu_a(a,m,\theta)}{\partial\theta_{ma}} \,, \\ \frac{\partial\mu(a,o,\theta)}{\partial\theta_{mom'}} &= \frac{\partial\mu_m(m,o,m',\theta)}{\partial\theta_{mom'}} \,.\end{aligned} \tag{2.5}$$

The derivation of the parameter update becomes identical to that of section 2.2. The resulting algorithm[2] is given in table 2.3 (compare to

Tab. 2.3: The GAPS algorithm for FSC with a look-up table.

```
**Initialize policy: For** all $(o, a)$: $\theta_{oa} \leftarrow 0$, $\theta_{mom'} \leftarrow 0$
**For** $i = 1$ to $n$: (make $n$ trials)
  • Beginning of the trial:
      $R \leftarrow 0$
      Initialize controller state $m(0)$
      for all $(o, a, m, m')$:
          $N_m \leftarrow 0$, $N_{ma} \leftarrow 0$, $N_{mom'} \leftarrow 0$, $N_{mo} \leftarrow 0$
  • At each time step $t$ of the trial:
      inc$(N_m)$
      Get observation $o(t)$, inc$(N_{mo})$
      Draw next $m(t)$ from $\mu_m(m(t), o(t), m(t-1), \theta)$
      inc$(N_{mom'})$
      Draw next $a(t)$ from $\mu_a(a(t), m(t), \theta)$
      inc$(N_{ma})$
      Execute $a(t)$, get reward $r(t)$
      $R \leftarrow R + \gamma^t r(t)$
  • End of the trial-update:
      for all $(o, a, m, m')$:
          $\theta_{ma} \leftarrow \theta_{ma} + \alpha R\,(N_{ma} - \mu_a(a, m, \theta) N_m)$
          $\theta_{mom'} \leftarrow \theta_{mom'} + \alpha R\,(N_{mom'} - \mu_m(m, o, m', \theta) N_{mo})$
```

table 2.2) and is based on following update equations:

$$\begin{aligned}\Delta\theta_{ma}(t) &= -\frac{\alpha}{\zeta}\gamma^t r^t\,(N_{ma} - \mu_a(a, m, \theta) N_m)\ ,\\ \Delta\theta_{mom'}(t) &= -\frac{\alpha}{\zeta}\gamma^t r^t\,(N_{mom'} - \mu_m(m, o, m', \theta) N_{mo})\ ,\end{aligned}$$

where the agent maintains the following counters of events up until time t: N_m is the number of times the controller has been in state m; N_{ma} is the number of times that action a has been executed while the controller has been in state m; N_{mo} is the number of times that observation o has been made while the controller has been in state m; $N_{mom'}$ is the number of times that the controller moved from state m to state m' after observation o.

Remark **2.6.1** *A lookup table with Boltzmann law is not the only option in the policy encoding. A simpler representation would be such that* $\mu_a(m,a) = \theta_{ma}$ *and* $\mu_m(m,o,m') = \theta_{mom'}$ *. Then the contribution to the update of each parameter at each time-step in the sequence can be expressed through same counters as:*

$$\begin{aligned} \Delta\theta_{ma}(t) &= -\alpha\gamma^t r^t \frac{N_{ma}(t)}{\theta_{ma}} , \\ \Delta\theta_{mom'}(t) &= -\alpha\gamma^t r^t \frac{N_{mom'}(t)}{\theta_{mom'}} . \end{aligned}$$

Despite its simplicity, this straightforward look-up table representation has several drawbacks. First, the parameters θ *represent probabilities, and thus they should belong to* $[0,1]$ *interval and add up to* 1*. However nothing guarantees that these constraints will remain satisfied as the agent applies the update rule described above. One way around this problem is forcing parameters back onto simplex by projecting the gradient onto the simplex before applying updates. Another issue is that some parameters could be set to zero and there will be no way to correct this, since the corresponding action will never be sampled. Studying how to express the gradient in such cases falls beyond the scope of this work (see [104]).*

The following section illustrates the performance of the GAPS algorithm with memory as FSC, outlined in table 2.3 by empirical results for partially observable pole balancing domain, in which the optimal policy requires several internal states.

2.7 Empirical Study of FSCs

In our experiments, we use the Boltzmann law (see equation 2.3) to represent the parameters of the graphs. This representation avoids both problems of look-up tables mentioned in remark 2.6.1: the weights can take any real values, and the induced policy never gives probability 0 to any choice. Note that the use of the Boltzmann law may alter the landscape of the function on which we perform gradient descent.

[2] This algorithm follows the credit assignment of non-episodic task as discussed in remark 2.1.1. It corresponds to using the immediate reward (error e_{policy} in VAPS [104]).

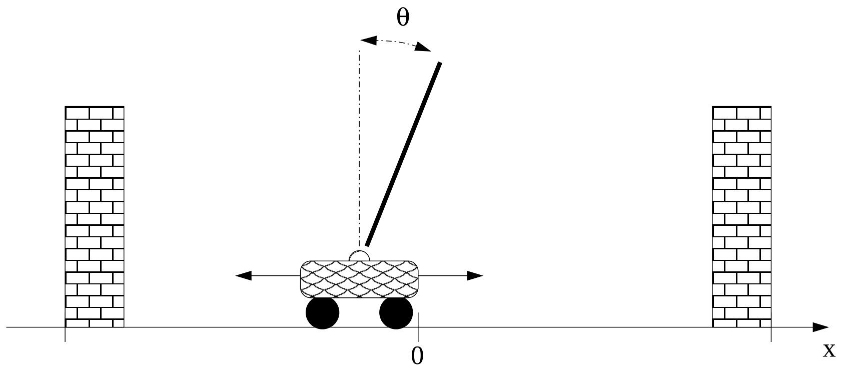


Fig. 2.6: The pole and cart balancing system.

Pole balancing We ran a number of experiments with the pole balancing problem [4, 167] illustrated in figure 2.6. The goal in this problem is to learn to avoid the cart hitting the ends of the track or the angle of the pole exceeding twelve degrees from the vertical. The specifications for the simulated pole and cart are: length of the track: 4.8 m; mass of the cart: $m_c = 1.0$ kg; the pole is 1 meter long and weights $m_p = .1$ kg; distance of pole's centre of mass from pivot: $l = 0.5$m; the applied control force $F = +10$ or -10 N. Let us denote by y the pole's angle (in radians), by x the cart's position on the track, by $m = m_c + m_p$ the mass of cart and pole together, by $g = 9.8\frac{m}{s^2}$ the gravity acceleration. The equations of motion were derived by Anderson [4]:

$$\ddot{y} = \frac{mg\sin y - \cos y(F + m_p l\dot{y}^2 \sin y)}{(4/3)ml - m_p l \cos^2 y}$$

$$\ddot{x} = \frac{[F + m_p l(\dot{y}^2 \sin y - \ddot{y}\cos y)]}{m},$$

Control actions are applied at the rate of 50Hz, e.g. 3000 consecutive control actions correspond to one minute of balancing time. Other parameters of the cart and pole balancing problem were taken as described in the supplementary WWW page for Sutton & Barto's book [167]. This well known problem is known to be solvable by a reactive policy (RP), if the observation at each time-step is composed of four elements: the cart position x and velocity $\dot{x}$, and the pole angle y and angular velocity $\dot{y}$.

The performance of learning algorithms was measured in two different settings. In a completely observable setting all four relevant variables x, $\dot{x}$, y and $\dot{y}$ were provided as the input to algorithms at each time-step. In a partially observable setting both $\dot{x}$ and $\dot{y}$ were hidden. Three algorithms applied in these two settings were: SARSA, Baird and Moore's original VAPS (learning an RP) and GAPS on FSCs with varying number of internal states.

SARSA and the original VAPS can be expected to succeed in the completely observable setting, and to fail in the partially observable one where there is no reactive policy that performs the task. These two algorithms differ radically. On one hand we use VAPS with the immediate error e_{policy} which makes it equivalent to TD(1); Baird and Moore would call this a pure policy search. On the other hand SARSA is basically a value-search similar to TD(0). The GAPS algorithm can be expected to succeed in both settings, provided that we use a sufficiently large FSC, and that the algorithm does not get stuck on a local optimum.

Two internal states should be enough in the completely observable setting, since every reactive policy using only two actions (as it is the case here) can be represented by a two-state FSC. In the partially observable framework, more internal states must be added to allow the algorithm to remember past observations.

In all experiments the discount factor γ was set to .99 and increased gradually as learning progressed. The learning rate α was optimized independently for each algorithm. The performance of the algorithm was measured by fixing the policy and executing 200 trials, measuring the length of each trial in terms of control decisions, and averaging these measures. The value intervals of cart position and pole position were partitioned into 6 and 3 unequal parts (smaller size of partition towards the center) in the completely observable setting, and into 8 and 6 parts in the partially observable setting, correspondingly. We were making decisions at the rate of 50 Hz, meaning, for example, that the actual physical time of learning to balance a pole for 500 sequential ticks corresponds to 10 seconds of balancing.

Figure 2.7 presents the learning curves obtained in the completely observable framework. The horizontal axis represents the number of trials, which corresponds to the number of times we have dropped the pole. The vertical axis represents the performance of the algorithm, measured as explained above. "RP" stands for the original VAPS al-

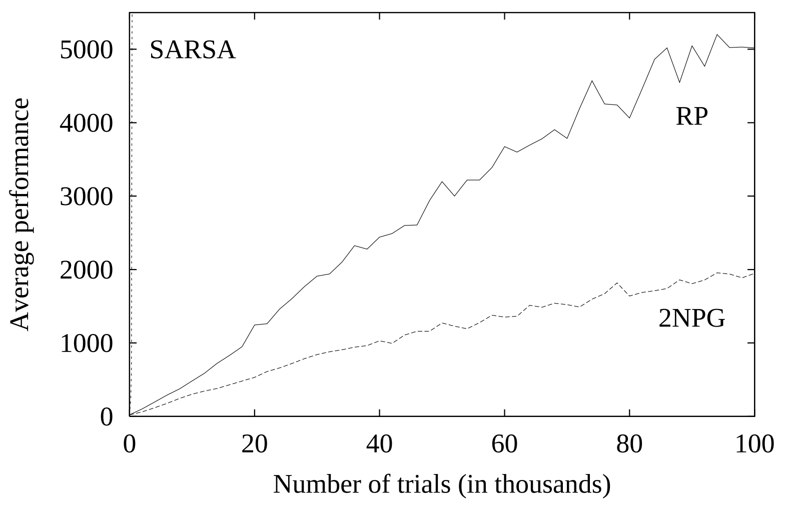


Fig. 2.7: Learning curves for the completely observable pole-balancing domain.

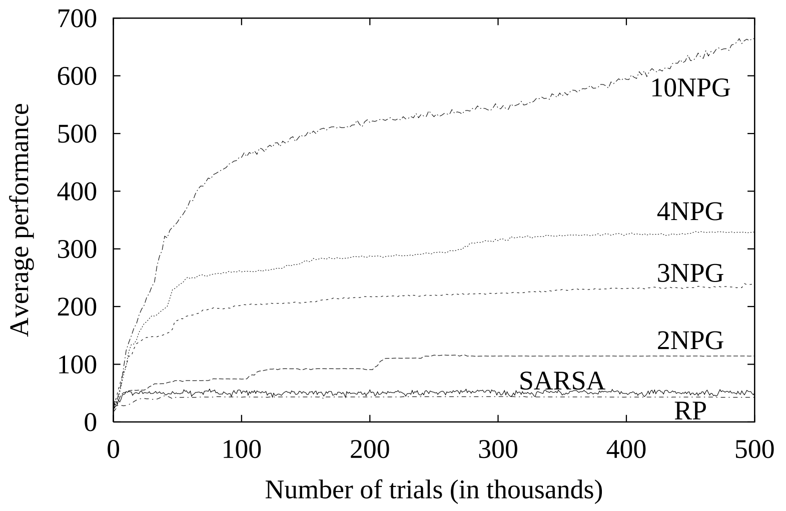


Fig. 2.8: Learning curves for the partially observable pole-balancing domain.

gorithm; "2NPG" represents GAPS used with N = 2. We see that: SARSA learns much faster than the original VAPS[3], showing that value search is much more efficient that policy search for this control problem; GAPS with 2-state FSC learns more slowly than the original VAPS. This phenomenon can be explained by the fact that the space of 2-state FSCs is bigger than the space of RPs.

Figure 2.8 presents the results obtained in the partially observable framework. "RP" stands for the original VAPS algorithm; "2NPG", "3NPG" , "4NPG" and "10NPG"represent GAPS using 2, 3, 4 and 10 internal states respectively. Note the difference in scales of these two figures. These results confirm our expectation that algorithms limited to reactive policies will fail. In contrast, our algorithm increases its performance gradually, showing it is able to compensate for the lack of observability. The more internal states are given to the algorithm, the better it performs. It is also striking to see that the performance of the algorithm seems to improve in steps, which makes difficult to predict where learning will stop.

We did not continue the experiments beyond 500,000 iterations, so we do not know whether the performance would continue to increase until the system could balance infinitely long. The most significant current result is that we can learn the structure of the FSC that extracts some useful information contained in the string of past observations, to compensate, at least partially, for the lack of observability.

Pole balancing is a widely accepted benchmark problem for dynamic system control and to the best of our knowledge it has not been learned with partial information with the exception of work by Mikkulainen *et al.* [107, 55]. Their work differs from ours in several principal aspects which make it impossible to compare results. Their approach is to look for the optimal controller in the space of recurrent neural networks, by means of evolution of the network parameters. The controller outputs a continuous value of force applied to a cart which supports two poles of different length. Experiments result in the size of maintained population and number of generations and offspring.

[3]Corresponds to a barely noticeable dashed line right next to the vertical axis.

Discussion

We have considered various ways of combining the basic gradient ascent algorithm for policy search with memory structures, necessary to act optimally in partially observable environments. Empirical results suggest that these algorithms are capable of learning small POMDPs, but they might not scale well. A possible direction for further research would be to develop memory structure and learning algorithms in order to take advantage of hierarchical structure in the domain and enable generalization. One way to do that would be to separate learning into distinct stages of learning reactive policies on small instances of domains and keeping these as a basis for learning in larger instances of same or similar domains. Admittedly, learning in this domain is very slow. It seems that these algorithms do not make good use of the information available since every experience is used only once to make an update, after which it is discarded. In certain constrained classes of domains and policies experience can be reused as we will demonstrate in chapter 5.

Chapter 3

Policy Search in Multiagent Environments

Summary At this point we turn a discussion to a multi-agent setting. This chapter examines the extension of previously introduced gradient-based algorithms to learning in cooperative games. Cooperative games are those in which all agents share the same payoff structure. For such a setting, there is a close correspondence between learning in a centrally controlled distributed system and in a system where components are controlled separately. A resulting policy learned by the distributed policy-search method for cooperative games is analyzed from the standpoint of both local optimum and Nash equilibrium—game-theoretic notions of optimality for strategies. The effectiveness of distributed learning is demonstrated empirically in a small, partially observable simulated soccer domain.

3.1 Cooperative Identical Payoff Games

The interaction of decision makers who share an environment is a complex issue that is traditionally studied by game theory and economics. The game theoretic formalism is very general, and analyzes the problem in terms of solution concepts such as Nash equilibrium [118], but usually works under the assumption that the environment is perfectly known to the agents.

In this chapter we describe a gradient-ascent policy-search algorithm for cooperative multi-agent domains. In this setting, after each agent performs its action given its observation according to some individual strategy, they all receive the same payoff. Our objective is to find a learning algorithm that makes each agent independently find a strategy that enables the group of agents to receive the optimal payoff. Although this will not be possible in general, we present a distributed algorithm that finds *local* optima in the space of the agents' policies.

An identical payoff stochastic game (IPSG) [110] describes the interaction of a set of agents with a Markov environment. That is, the state of the environment at time t depends only on its previous state and the actions of the agents. Formally, an *identical payoff stochastic game* (IPSG)[1] is a tuple $\langle S, \pi_0^S, G, T, \rho \rangle$, where

- S is a discrete state space;
- π_0^S is a probability distribution over the initial state;
- G is a collection of agents, where an *agent* i is a 3-tuple of $\langle A^i, O^i, B^i \rangle$
  - its discrete action space A^i,
  - discrete observation space O^i, and
  - observation function $B^i \colon S \to \mathcal{P}(O^i)$;
- $\mathcal{T} : S \times \mathcal{A} \to \mathcal{P}(S)$ is a mapping from states of the environment and actions of the agents to probability distributions over states of the environment; and
- $\rho : S \times \mathcal{A} \to \mathcal{R}$ is the payoff function, where $\mathcal{A} = \prod_i A^i$ is the joint action space of the agents.

Because all players receive the same payoffs, it is called an *identical payoff game*.

When all agents in G have the identity observation function $B(s) = s$ for all $s \in S$, the game is *completely observable*. Otherwise, it is a *partially observable* IPSG (POIPSG). POIPSGs are often a natural model

[1]IPSG's are also called *stochastic games* [66], Markov games [87] and *multi-agent Markov decision processes* [24].

for multi-agent systems in which the agents are in different physical locations, observing only part of the system state, dependent on their location. Figure 3.1 depicts an influence diagram for variables describing two agents controlled by FSCs interacting with an environment. In a POIPSG, at each time step:

- each agent $i \in (1..k)$ observes $o_i(t)$ corresponding to $B^i(s(t))$ and selects an action $a_k(t)$ according to its strategy;
- a compound action $\vec{a}(t) = (a_1(t), \ldots, a_k(t))$ from the joint action space $\mathcal{A}$ is performed, inducing a state transition of the environment; and
- the identical reward $r(t)$ is received by all agents.

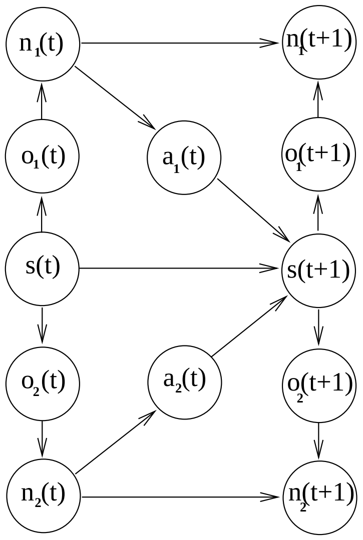


Fig. 3.1: An influence diagram for two agents with FSCs in POMDP.

The objective of each agent is to choose a strategy that maximizes the *value of the game*. For a discount factor $\gamma \in [0, 1)$ and a set of strategies $\vec{\theta} = (\theta_1, \ldots, \theta_k)$, given the distribution over initial state of the environment π_0^S, the value of the game is

$$V(\vec{\theta}, \pi_0^S) = \sum_{t=0}^{\infty} \gamma^t E(r(t) \mid \vec{\theta}, \pi_0^S).$$

Note that in a completely observable IPSG, reactive policies are sufficient to implement the best possible joint strategy. This follows directly from the fact that every MDP has an optimal deterministic reactive policy [133]. Therefore an MDP with the product action space $\prod_i A^i$ corresponding to a completely observable IPSG also has one. This deterministic reactive policy is representable by deterministic reactive policies for each agent. Moreover, it has been shown that in partially observable environments, the best memoryless policy can be arbitrarily worse than the best policy using memory [157]. This statement can also be easily extended to IPSGs. There are many possibilities for constructing policies with memory [123, 104]. In this work we use a *finite state controller* (FSC) for each agent. A description of FSCs is given in Section 2.5 (also see the paper by Meuleau *et al.* [104]).

To better understand IPSGs, let us consider an example illustrated in figure 3.2 (adopted from Boutilier [24]). There are two agents, a_1 and a_2, each of which has a choice of two actions, a and b, at any of three states. All transitions are deterministic and are labeled by the joint action that corresponds to the transition. For instance, the joint action (a, b) corresponds to the first agent performing action a and the second agent performing action b. Here, $*$ refers to any action taken by the corresponding agent. The starting state is $s1$, where the first agent alone decides whether to move the environment to state $s2$ by performing action a or to state $s3$ by performing action b. In state $s3$, no matter what both agents do as the next step, they receive a reward of $+5$ in state $s6$ risk-free. In state $s2$, the agents have a choice of cooperating—choosing the same action, whether (a, a) or (b, b)—with reward $+10$ in state $s4$, or not—choosing different actions, whether (a, b) or (b, a)—and getting -10 in state $s5$. We will represent a joint policy with parameters p^{Agent}_{State}, denoting the probability that an agent will perform action a in the corresponding state. Only three parameters are important for the outcome: $\{p^1_1, p^1_2; p^2_2\}$. The optimal joint policies are $\{1, 1; 1\}$ or $\{1, 0; 0\}$, which are deterministic reactive policies.

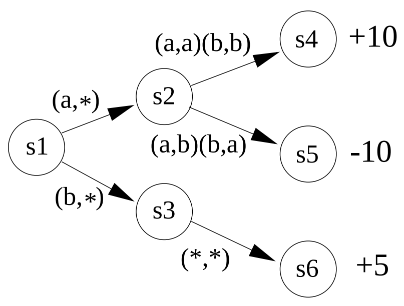


Fig. 3.2: A coordination problem in a completely observable identical payoff game.

3.2 GAPS in Multiagent Environments

In this section, we show how to apply a general method for using gradient ascent in policy spaces to multi-agent problems. We compare the case of centralized control of a system with distributed observations and actions to that of distributed control.

Central control of factored actions

Let us consider the case in which the action is factored, meaning that each action $\vec{a}$ consists of several components $\vec{a} = (a_1, \ldots, a_k)$. We can consider two kinds of controllers: a *joint controller* is a policy mapping observations to the complete joint distribution $\mathcal{P}(\vec{a})$; a *factored controller* is made up of independent sub-policies $\theta_{a_i} : O^i \to \mathcal{P}(a_i)$ (possibly with a dependence on individual internal state) for each action component.

Factored controllers can represent only a subset of the policies represented by joint controllers. Obviously, any product of policies for the factored controller $\prod_i \theta_{a_i}$ can be represented by a joint controller $\theta_{\vec{a}}$, for which $\Pr(\vec{a}) = \prod_{i=1}^{k} \Pr(a_i)$. However, there are some stochastic joint controllers that cannot be represented by any factored controller, because they require coordination of probabilistic choice across action components, which we illustrate by the following example.

Remark **3.2.1** *The first action component controls the liquid component of a meal* $a_1 \in \{\text{vodka}, \text{milk}\}$ *and the second controls the solid one* $a_2 \in \{\text{pickles}, \text{cereal}\}$. *Let us assume that that for a healthy diet, we sometimes want to eat milk with cereal, other times vodka with pickles. The optimal policy is randomized, say* 10% *of the time* $\vec{a} = (vodka, pickles)$ *and* 90% *of the time* $\vec{a} = (milk, cereal)$. *But when the liquid and solid components are controlled independently, one cannot represent this policy. With randomization, we are occasionally forced to have vodka with cereal or milk with pickles.*

Because we are interested in individual agents learning independent policies, we concentrate on learning the best factored controller for a domain, even if it is suboptimal in a global sense. Requiring a controller to be factored simply puts constraints on the class of policies, and therefore distributions $P(a \mid \theta, h)$, that can be represented. The stochastic gradient ascent techniques of the previous section can still be applied directly in this case to find local optima in the factored controller space. We will call this method *joint gradient ascent.* For the case of FSCs we are going to assume a disjoint encoding of the action function and the internal state transition function. A particular case would be a look-up table encoding (see equations (2.5)).

Distributed control of factored actions

The next step is to learn to choose action components not centrally, but under the distributed control of multiple agents. One obvious strategy would be to have each agent perform the same gradient-ascent algorithm in parallel to adapt the parameters of its own local policy θ_{a_i}. Perhaps surprisingly, this *distributed gradient ascent policy search* (DGAPS) method is very effective.

Theorem 3.1 *In partially observable identical payoff stochastic game, for factored controllers, distributed gradient ascent is equivalent to joint gradient ascent.*

Proof: We will show that for both controllers the algorithm will be stepwise the same, so starting from the same point in the search space, on the same data sequence, the algorithms will converge to the same locally optimal parameter setting. For a factored controller, a joint experience $\vec{h}$ can be described as

$$\langle m_1(0), ...m_k(0), o_1(1), ...o_k(1), m_1(1), ...m_k(1), a_1(1), ...a_k(1), r(1), ...\rangle$$

and the corresponding experience h_i for an individual agent i is

$$\langle m_i(0), o_i(1), m_i(1), a_i(1), r(1), ...\rangle \ .$$

It is clear that a collection $h_1...h_k$ of individual experiences, one for each agent, specifies the joint experience $\vec{h}$. In what follows, we are going to present analysis for parameters θ_{ma} and leave out identical analysis for θ_{mom}. The joint gradient ascent algorithm requires that we draw sample experiences from $\Pr(\vec{h}|\pi_0^S, \vec{\theta})$ and that we do gradient ascent on $\vec{\theta}$ with a sample of the gradient (see equation (2.2)) at each time step t in the experience h equal to

$$\gamma^t r(t,h) \sum_{\tau=1}^{t} \frac{\partial \ln \Pr(a(\tau,h) \mid \pi_0^S, \vec{h}^{\tau-1}, \theta)}{\partial \theta_{ma}} \ .$$

Whether a factored controller is being executed by a single agent, or it is implemented by agents individually executing policies θ_{a_i} in parallel, joint experiences are generated from the same distribution $\Pr(\vec{h} \mid \pi_0^S, \langle \theta_{a_1}, ..., \theta_{a_k} \rangle)$. So the distributed algorithm is sampling from the correct distribution.

Now, we must show that the weight updates are the same in the distributed algorithm as in the joint one. Let $\vec{\theta}_w = (\theta_w^0, \ldots, \theta_w^{M_w})$ be the set of parameters controlling action component a_w. Then

$$\frac{\partial}{\partial \theta_w^j} \ln(\Pr(a_l(\tau) \mid \pi_0^S, \vec{h}_l^{\tau-1}, \theta_l)) = 0 \text{ for all } w \neq l;$$

that is, the action probabilities of agent l are independent of the parameters in other agents' policies. With this in mind, for factored controllers, the derivative in expression 2.2 becomes

$$\begin{aligned}
&\frac{\partial}{\partial \theta_w^j} \ln \Pr\left(\vec{a}(\tau, h) \mid \pi_0^S, \vec{h}^{\tau-1}, \vec{\theta}\right) \\
&\qquad = \frac{\partial}{\partial \theta_w^j} \ln \prod_{i=1}^{k} \Pr\left(a_i(\tau, h) \mid \pi_0^S, h_i^{\tau-1}, \theta_i\right) \\
&\qquad = \sum_{i=1}^{k} \frac{\partial}{\partial \theta_w^j} \ln \Pr\left(a_i(\tau, h) \mid \pi_0^S, h_i^{\tau-1}, \theta_i\right) \\
&\qquad = \frac{\partial}{\partial \theta_w^j} \ln \Pr\left(a_w(\tau, h) \mid \pi_0^S, h_w^{\tau-1}, \theta_w\right) \ .
\end{aligned}$$

Therefore, the same weight updates will be performed by distributed GAPS as by joint gradient ascent on a factored controller. □

This theorem shows that policy learning and control over component actions can be distributed among independent agents who are not aware of each others' choice of actions. An important requirement, though, is that agents perform simultaneous learning (which might be naturally synchronized by the coming of the rewards).

3.3 Relating Local Optima in Policy Space to Nash Equilibria

In game theory the Nash equilibrium is a common solution concept. Because gradient ascent methods can often be guaranteed to converge to local optima in the policy space, it is useful to understand how those points are related to Nash equilibria. We will limit our discussion to the two-agent case, but the results are generalizable to more agents.

A Nash equilibrium is a pair of strategies such that deviation by one agent from its strategy, assuming the other agent's strategy is fixed, cannot improve the overall performance. Formally, in an IPSG, a *Nash equilibrium* point is a pair of strategies (θ_1^*, θ_2^*) such that:

$$V(\langle\theta_1^*, \theta_2^*\rangle, \pi_0^S) \geq V(\langle\theta_1, \theta_2^*\rangle, \pi_0^S),$$
$$V(\langle\theta_1^*, \theta_2^*\rangle, \pi_0^S) \geq V(\langle\theta_1^*, \theta_2\rangle, \pi_0^S)$$

for all $\langle\theta_1, \theta_2\rangle$. When the inequalities are strict, it is called a *strict Nash equilibrium*.

Every discounted stochastic game has at least one Nash equilibrium point [66]. It has been shown that under certain convexity assumptions about the shape of payoff functions, the gradient ascent process converges to an equilibrium point [6]. It is clear that the optimal Nash equilibrium point (the Nash equilibrium with the highest value) in an IPSG also is a possible point of convergence for the gradient ascent algorithm, since it is the global optimum in the policy space.

Let us return to the game described in Figure 3.2. It has two optimal strict Nash equilibria at $\{1, 1; 1\}$ and $\{1, 0; 0\}$. It also has a set of sub-optimal Nash equilibria $\{0, p_2^1; p_2^2\}$, where p_2^2 can take on any value in the interval $(.25, .75)$ and p_2^1 can take any value in the interval $[0, 1]$. The sub-optimal Nash equilibria represent situations in which the first agent always chooses the bottom branch and the second agent acts moderately randomly in state $s2$. In such cases, it is strictly better for the first agent to stay on the bottom branch with expected value $+5$. For the second agent, the payoff is $+5$ no matter how it behaves, so it has no incentive to commit to a particular action in state $s2$ (which is necessary for the upper branch to be preferred).

In this problem, the Nash equilibria are also all local optima for the gradient ascent algorithm. Unfortunately, this equivalence only holds in one direction in the general case. We state this more precisely in the following theorems.

Theorem 3.2 *In partially observable identical payoff stochastic game, every strict Nash equilibrium is a local optimum for gradient ascent in the space of parameters of a factored controller.*

Proof: Assume that we have two agents and denote the strategy at the point of strict Nash equilibrium as (θ_1^*, θ_2^*) encoded by parameter vector $\langle\theta_1^1 \ldots \theta_1^G, \theta_2^1 \ldots \theta_2^G\rangle$. For simplicity, let us further assume that (θ_1^*, θ_2^*) is not on the boundary of the parameter space, and each weight is locally relevant: that is, that if the weight changes, the policy changes, too.

By the definition of Nash equilibrium, any change in value of the parameters of one agent without change in the other agent's parameters results in a decrease in the value $V()$. In other words, we have that $\partial V / \partial \theta_i^j \leq 0$ *and* $-\partial V / \partial \theta_i^j \leq 0$ for all j and i at the equilibrium point. Thus, $\partial V / \partial \theta_i^j = 0$ for all θ_i^j at (θ_1^*, θ_2^*), which implies it is a singular point of $V()$. Furthermore, because the value decreases in every direction, it must be a maximum.

In the case of a locally irrelevant parameter θ_i^j, $V()$ will have a ridge along its direction. All points on the ridge are singular and, although they are not strict local optima, they are essentially local optima for gradient ascent. □

The problem of Nash equilibria on the boundary of the parameter space is an interesting one. Whether or not they are convergence points depends on the details of the method used to keep gradient ascent within the boundary. A particular problem comes up when the equilibrium point occurs when one or more parameters have infinite value (this is not uncommon, as we shall see in section 3.4). In such cases, the equilibrium cannot be reached, but it can usually be approached closely enough for practical purposes.

Theorem 3.3 *For partially observable identical payoff stochastic game, some local optima for gradient ascent in the space of parameters of a factored controller are not Nash equilibria.*

Proof: From the point of view of a single agent, the theorem states that a local optimum need not be a global one. Consider a situation in which each agent's policy has a single parameter θ_i, so the policy space can be described by $\langle \theta_1, \theta_2 \rangle$. We can construct a value function $V(\theta_1, \theta_2)$ such that for some c, $V(\cdot, c)$ has two modes, one at $V(a, c)$ and the other at $V(b, c)$, such that $V(b, c) > V(a, c)$. Further assume that $V(a, \cdot)$ and $V(b, \cdot)$ each have global maxima $V(a, c)$ and $V(b, c)$. Then $V(a, c)$ is a local optimum that is not a Nash equilibrium. □

3.4 Experiments with Cooperative Games

According to our knowledge, there are no established benchmark problems for multi-agent learning. To illustrate our method we present empirical results for two problems: the simple coordination problem discussed earlier and illustrated in figure 3.2 and for a small multi-agent simulated soccer domain.

Simple coordination without communication

We originally discussed policies for this problem using three weights, one corresponding to each of the p_{State}^{Agent} probabilities. However, to force gradient ascent to respect the bounds of the simplex, we used the standard Boltzmann encoding, so that for agent i in state s there are two weights, $\theta_{s,a}^{i}$ and $\theta_{s,b}^{i}$, one for each action. The action probability is coded as a function of these weights as $p_s^i = \frac{\exp(\theta_{sa}^i)}{\exp(\theta_{sa}^i)+\exp(\theta_{sb}^i)}$.

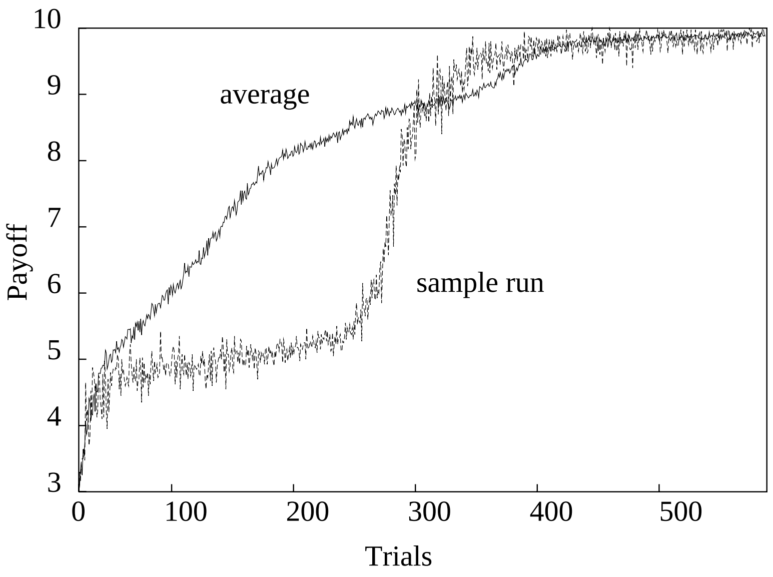


Fig. 3.3: Learning curve for the 6-state coordination problem.

Figure 3.3 shows the result of running distributed GAPS with a learning rate of $\alpha = .003$ and a discount factor of $\gamma = .99$. The graph of a sample run illustrates how the agents typically initially move towards a sub-optimal policy. The policy in which the first agent always takes action b and the second agent acts fairly randomly is a Nash equilibrium, as we saw in section 3.3. However, this policy is not exactly representable in the Boltzmann parameterization because it requires one of the weights to be infinite to drive a probability to either 0 or 1. So, although the algorithm moves toward this policy, it never reaches it exactly. This means that there is an advantage for the second agent to drive its parameter toward 0 or 1, resulting in eventual convergence toward a global optimum (note that, in this parameterization, these optima cannot be reached exactly, either). The average of ten runs shows that the algorithm always converges to a pair of policies with value very close to the maximum value of 10.

Soccer Domain

We have conducted experiments on a small soccer domain adapted from Littman [87]. The game is played on a 6×5 grid as shown in Figure 3.4. There are two learning agents on one team and a single opponent with a fixed strategy on the other. Every time the game begins, the learning agents are randomly placed in the right half of the field, and the opponent in the left half of the field. Each cell in the grid contains at most one player. Every player on the field (including the opponent) has an equal chance of initially possessing the ball. At each time step, one of the six actions: $\{North, South, East, West, Stay, Pass\}$ is executed by a player. When an agent passes, the ball is transferred to the other agent on its team on the next time step. Once all players have selected actions, they are executed in a random order. When a player executes an action that would move it into the cell occupied by some other player, possession of the ball goes to the stationary player and the move does not occur. When the ball falls into one of the goals, the game ends and a reward of ± 1 is given.

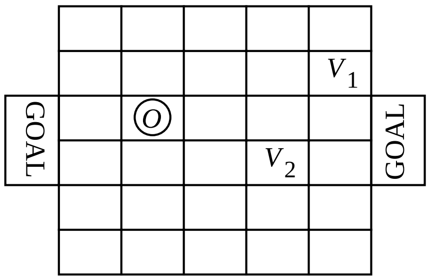


Fig. 3.4: The soccer field with two learning agents (V_1 and V_2) and the opponent (O).

We made a partially observable version of the domain to test the effectiveness of DGAPS: each agent can only obtain information about which player possesses the ball and the status of cells to the north, south, east and west of its location. There are three possible observations for each cell: whether it is open, out of the field, or occupied by someone.

Results for Soccer Domain

In figures 3.5, 3.6, 3.7 and 3.8 we compare the learning curves of DGAPS to those of Q-learning with a central controller for both the completely observable and the partially observable cases. We also show learning curves of DGAPS without the action *Pass* in order to measure the *cooperativeness* of the learned policies. The graphs summarize simulations of the game against three different fixed-strategy opponents:

- Random: Executes actions uniformly at random.
- Greedy: Moves toward the player possessing the ball and stays there. Whenever it has the ball, it rushes to the goal.
- Defensive: Rushes to the front of its own goal, and stays or moves at random, but never leaves the goal area.

We show the average performance over 10 runs. The learning rate was 0.05 for distributed GAPS and 0.1 for Q-learning, and the discount factor was 0.999, throughout the experiments. Each agent in the DGAPS team learned a four-state finite-state controller. The controller parameters

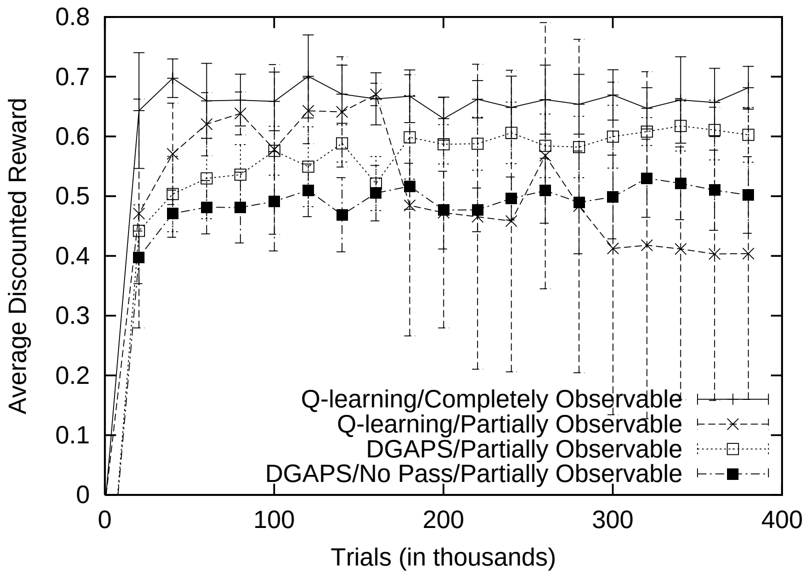


Fig. 3.5: Learning curve for a defensive opponent.

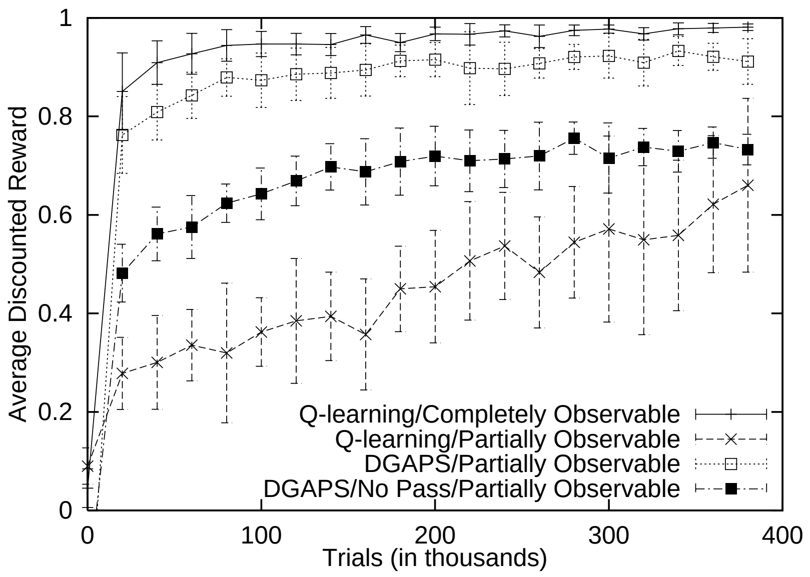


Fig. 3.6: Learning curve for a greedy opponent.

were initialized by drawing uniformly at random from the appropriate domains. We used ϵ-greedy exploration with $\epsilon = .04$ for Q-learning. The performance in the graph is reported by evaluating the greedy policy derived from the look-up table by picking the maximal Q-value.

Because, in the completely observable case, this domain is an MDP (the opponent's strategy is fixed, so it is not really an adversarial game), Q-learning can be expected to learn the optimal joint policy, which it seems to do. It is interesting to note the slow convergence of completely observable Q-learning against the random opponent. We conjecture that this is because, against a random opponent, a much larger part of the state space is visited. The table-based value function offers no opportunity for generalization, so it requires a great deal of experience to converge.

As soon as observability is restricted, Q-learning no longer reliably converges to the best strategy. The joint Q-learner now has as its input the two local observations of the individual players. It behaves quite erratically, with extremely high variance because it sometimes converges to a good policy and sometimes to a bad one. This unreliable

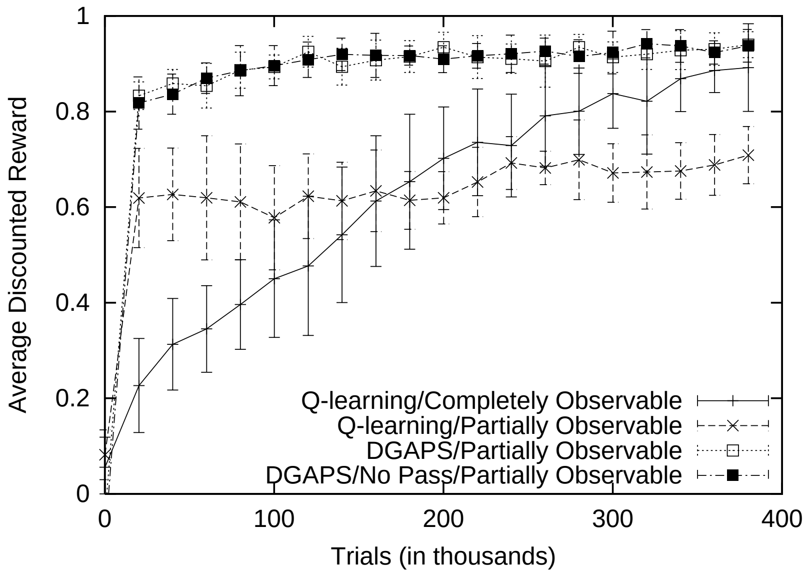


Fig. 3.7: Learning curve for a random opponent.

behavior can be attributed to the well-known problems of using value-function approaches, and especially Q-learning, in POMDPs.

The individual DGAPS agents have internal state as well as stochasticity in their action choices, which gives them some representational abilities unavailable to the Q-learner. However, because they have only four internal states, they are not able to represent a complete reactive policy, with a different action for each possible observation. This restriction forces generalization, but also has the potential to make it impossible to represent a good strategy.

Despite the fact that they learn independently, the combination of policy search plus a different policy class allows them to gain considerably improved performance. We cannot tell how close this performance is to the optimal performance with partial observability, because it would be computationally impractical to solve the POIPSG exactly. Bernstein *et al.* [19] show that in the finite-horizon case, two-agent POIPSGs are *harder* to solve than POMDPs (in the worst-case complexity sense).

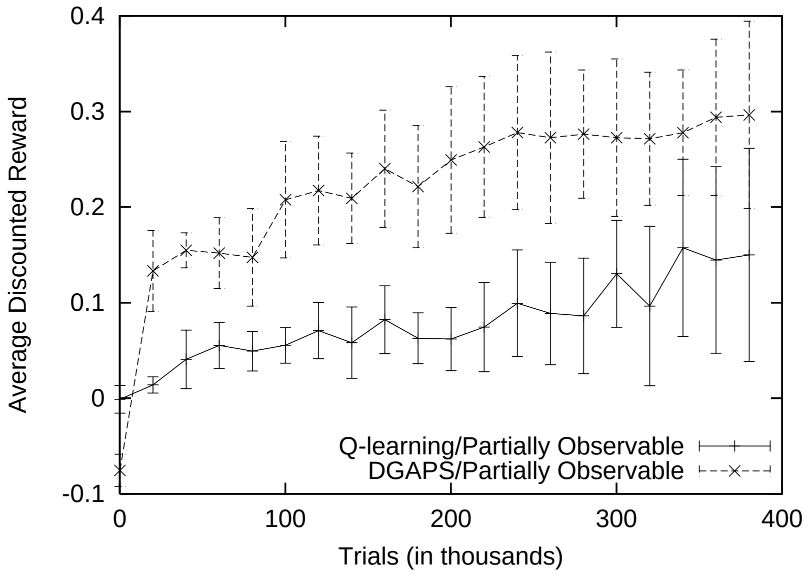


Fig. 3.8: Learning curve for a team of two agents.

It is also important to see that the two DGAPS agents have learned to cooperate in some sense: when the same algorithm is run in a domain without the "pass" action, which allows one agent to give the ball to its teammate, performance deteriorates significantly against both defensive and greedy opponents. Against a completely random opponent, both strategies do equally well. It is probably sufficient, in this case, to simply run straight for the goal, so cooperation is not necessary.

We performed some additional experiments in a two-on-two domain in which one opponent behaved greedily and the other defensively. In this domain, the completely observable state space is so large that it is difficult to even store the Q table, let alone populate it with reasonable values. Thus, we just compare two four-state DGAPS agents with a limited-view centrally controlled Q-learning algorithm. Not surprisingly, we find that the DGAPS agents are considerably more successful.

Finally, we performed informal experiments with an increasing number of opponents. The opponent team was made up of one defensive agent and an increasing number of greedy agents. For all cases in which the opponent team had more than two greedy agents, DGAPS led to a defensive strategy in which, most of the time, all agents rushed to the front of their goal and remained there.

3.5 Related Work

There is a wide interest in applying reinforcement learning algorithms to multi-agent environments. For example, Littman [87] describes and analyzes a *Q-learning*-like algorithm for finding optimal policies in the framework of zero-sum Markov games, in which two players have strictly opposite interests. Hu and Wellman [66] propose a different multi-agent Q-learning algorithm for *general-sum* games, and argue that it converges to a Nash equilibrium.

A simpler, but still interesting case, is when multiple agents share the same objectives. A study of the behavior of agents employing Q-learning individually was made by Claus and Boutilier [37], focusing on the influence of game structure and exploration strategies on convergence to Nash equilibria. In Boutilier's later work [24], an extension of value iteration was developed that allows each agent to reason explicitly about the state of coordination.

However, all of this research uses *value search* methods that are appropriate in completely observable Markov environments, but not necessarily useful in partially observable environments. It is quite natural in multi-agent systems to assume that when the different agents are in different physical locations, they do not observe each others' states.

Assuming that parts of the distributed system are able to communicate locally and that the reinforcement signal can be somehow apportioned to local components, Schneider *et al.* [148] proposed a value learning rule which was empirically found to be effective for some applications, but in general produces a solution with uncertain properties. Bartlett and Baxter [11] have independantly arrived to conclusions very similar to ours [122, 125] for distributed gradient ascent for poliy search. They stress biological plausibility of this algorithms for synaptic update rule in spiking neurons that involves only local quantities and reward signal, as opposed to e.g. back-propagation of error algorithms [140, 139].

3.6 Discussion

We have presented an algorithm for distributed learning in cooperative multi-agent domains. It is guaranteed to find local optima in the space of factored policies. We cannot show, however, that it always converges to a Nash equilibrium, because there are local optima in policy space that are not Nash equilibria. The algorithm performed well in a small simulated soccer domain. It is not guaranteed that in more complex domains the gradient remains strong enough to drive the search effectively and numerous local optima are not problematic. Chapter 4 presents an application of distributed GAPS to a complex domain of adaptive routing.

An interesting extension of this work would be to allow the agents to perform explicit communication actions with one another to see if they are exploited to improve performance in the domain. The performance of the algorithm itself might be improved through more careful exploration strategies coupled with the methods for experience reuse, described in chapter 5. It seems that care must be taken to ensure that only one agent is exploring at a time. Finally, there may be more interesting connections to establish with game theory, especially in relation to solution concepts other than Nash equilibrium, which may be more appropriate in cooperative games.

Chapter 4

Adaptive Routing

Summary In this chapter, we will apply the RL algorithm developed earlier to network routing. Successful telecommunications require efficient resource allocation which can be achieved by developing adaptive control policies. Reinforcement learning presents a natural framework for the development of such policies by trial and error in the process of interaction with the environment. Effective network routing means selecting the optimal communication paths. It can be modeled as a multi-agent RL problem and solved using the distributed gradient ascent algorithm. Performance of this method is compared to that of other algorithms widely accepted in the field. Conditions in which our method is superior are presented.

4.1 Description of Routing Domain

Successful telecommunications requires efficient resource allocation which can be achieved by developing adaptive control policies. Reinforcement learning presents a natural framework for the development of such policies by trial and error in the process of interaction with the environment. In a sense, learning an optimal controller for network routing can be thought of as learning in some episodic task of a kind we have seen in earlier chapters, like maze searching or pole balancing, but repeating trials many times in parallel with interaction among trials. Under this interpretation, an individual router is an agent which makes its routing decisions according to an individual policy. The parameters of this policy are adjusted according to some measure of the global performance of the network, while control is determined by local observations.

The approach described in chapter 3 allows to update the local policies, avoiding the necessity for centralized control or global knowledge of the networks structure. The only global information required by the learning algorithm is the network utility expressed as a reward signal distributed once an epoch (corresponding to the successful delivery of some predefined number of packets) and dependent on the average routing time.

We test our algorithm on a domain adopted from Boyan and Littman [25]. It is a discrete time simulator of communication networks with various topologies and dynamic structure. A communication network is an abstract representation of real-life systems such as the Internet or a transport network. It consists of a homogeneous set of nodes and edges between them representing links (see Figure 4.1). Nodes linked to each other are called neighbors. Links may be active ("up") or inactive ("down"). Each node can be the origin or the final destination of packets, or serve as a router.

Packets are periodically introduced into the network with a uniformly random node of origin and destination. They travel to their destination node by hopping between intermediate nodes. No packets are generated being destined to the node of origin. Sending a packet down a link incurs a cost that can be thought of as time in transition. There is an added cost to waiting in the queue of a particular node in order to access the router's computational resource—a queue delay. Both costs are assumed to be uniform throughout the network. In our experiments, each is set to be a unit cost. The level of network traffic is determined by the number of packets in the network. Once a packet reaches its destination, it is removed. If a packet has made significantly more hops than the total number of nodes in the network, it is also removed as a hopeless case. Multiple packets line up at nodes in an FIFO (first in first out) queue limited in size. The node must forward the top packet in the FIFO queue to one of its neighbors.

In the terminology of RL, the network is the environment whose state is determined by the number and relative position of nodes, the status of links between them, and the dynamics of packets. The destination of the packet and the status of local links form the node's observation. Each node is an agent who has a choice of actions. It decides where to send the packet according to a policy. Just like any policy, a routing policy could be deterministic or stochastic. A policy computed by our algorithm is stochastic, which means it sends a packet to the same destination down different links according to some distribution. The policy considered in our experiments does not decide whether or not to accept a packet (admission control), how many packets to accept from each neighbor, or which packets should be assigned priority.

The node updates the parameters of its policy based on the reward. Its reward comes in the form of a signal distributed through the network once the packet has reached its final destination. The reward depends on the total delivery time for the packet. We measure the performance of the algorithm by the average delivery time for packets once the system has settled on a policy.

Each packet is assumed to carry some elements of its routing history in adition to obvious destination and origin information. They include the time when the packet was generated, the time the packet last received attention from some router, the trace of recently visited nodes and the number of hops performed so far. In case a packet is detected to have spent too much time in the network failing to reach its destination, such packet is discarded and the network is penalized accordingly. Thus, a defining factor in our simulation is whether the number of hops performed by a packet is more than the total number of nodes in the network.

4.2 Algorithmic Details

We compare the performance of our distributed GAPS algorithm to three others, as follows. "Best" is a static routing scheme, based on the shortest path counting each link as a single unit of routing cost. We include this algorithm because it provides the basis for most current industry routing heuristics [18, 45]. "Bestload"performs routing according to the shortest path while taking into account queue sizes at each node. It is close to the theoretical optimum among deterministic routing algorithms even though the actual best possible routing scheme requires not simply computing the shortest path based on network loads, but also analyzing how loads change over time according to routing decisions. Since calculating the shortest path at every single step of the simulation would be prohibitively costly in terms of computational resources, we implemented "Bestload" by readjusting the routing policy only after a notable change of loads in the network. We consider 50 successfully delivered packets to constitute a notable load change. Finally, "Q-routing" is a distributed RL algorithm applied specifically to this domain by Littman and Boyan [25]. While our algorithm is stochastic and performs policy search, Q-routing is a deterministic, value search algorithm. Note that our implementation of the network routing simulation is based on the software Littman and Boyan used to test Q-routing. Even so, the results of our simulation of "Q-routing" and "Best" on the "6x6" network differ slightly from Littman and Boyan's due to certain modifications in traffic modeling conventions. For instance, we consider a packet delivered and ready for removal only after it has passed through the queue of the destination node and accessed its computational resources, and not merely when the packet is successfully routed to the destination node by an immediate neighbor, as in the original simulation.

We undertake the comparison between GAPS and the aforementioned algorithms with one important caveat. The GAPS algorithm explores the class of stochastic policies while all other methods pick deterministic routing policies. Consequently, it is natural to expect GAPS to be superior for certain types of network topologies and loads, where the optimal policy is stochastic. Later, we show that our experiments confirm this expectation.

We implement the distributed GAPS in POMDP. In particular, we represent each router as a POMDP, where the state contains the sizes of all queues, the destinations of all packets, the state of links (up or down); the environment state transition function is a law of the dynamics of network traffic; an observation o consists of the destination of the packet; an action a corresponds to sending the packet down a link to an adjacent node; and finally, the reward signal is the average number of packets delivered per unit of time. A routing policy in GAPS is represented by a look-up table. Each agent uses a GAPS RL algorithm to move parameterization values down the gradient of the average reward. It has been shown [122] that an application of distributed GAPS causes the system as a whole to converge to a local optimum under stationarity assumptions. This algorithm is essentially the one described in chapter 3.

Policies were initialized in two different ways: randomly and based on shortest paths. We tried initialization with random policies uniformly chosen over the parameter space. With such initialization results are very sensitive to the learning rate.A high learning rate often causes the network to stick in local optima in combined policy space, with very poor performance. A low learning rate results in slow convergence. What constitutes high or low learning rate depends on the specifics of each network and we did not find any satisfactory heuristics to set it. Obviously, such features as average number of hops necessary to deliver a packet under the optimal policy as well as learning speed crucially depend on the particular characteristics of each network, such as number of nodes, connectivity and modularity.

These considerations led us to a different way of initializing controllers. Namely, we begin by computing the shortest path and set the controllers to route most of the traffic down the shortest path, while occasionally sending a packet to explore an alternative link. We call this "ϵ-greedy routing". In our experiments, ϵ is set to .01. We believe that this parameter would not qualitatively change the outcome of our experiments since it only influences exploratory behaviour in the beginning.

The exploration capacity of the algorithm is regulated in a different way as well. Temperature and learning rate are simply kept constant both for considerations of simplicity and for maintaining the controllers' ability to adjust to changes in the network, such as links failure. However, our experiments indicate that having a schedule for reducing learning rate after a key initial period of learning would improve performance. Alternatively, it would be interesting to explore different learning rates for the routing parameters on one hand, and the encoding of topological features on the other.

4.3 Empirical Results

We compared the routing algorithms on several networks with various number of nodes and degrees of connectivity and modularity, including the 116-node "LATA" telephone network. On all networks, the GAPS algorithm performed comparably or better than other routing algorithms. To illustrate the principal differences in the behavior of algorithms and the key advantages of distributed GAPS, we concentrate on the analysis of two routing problems on networks which differ in a single link location.

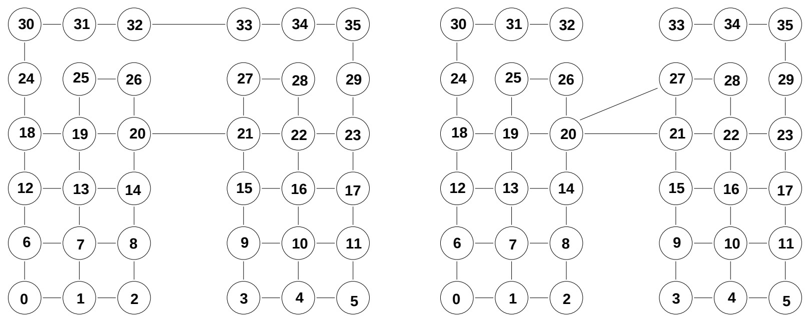
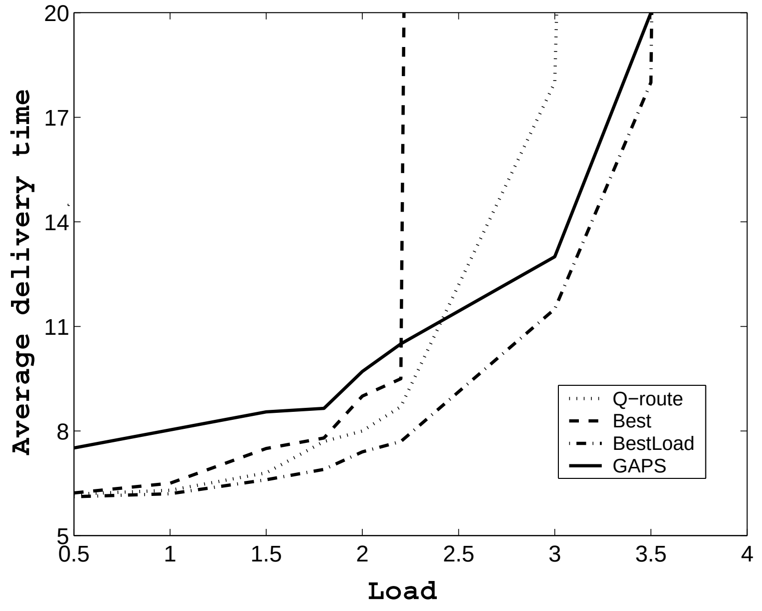


Fig. 4.1: The irregular grid topology.

Figure 4.1.left presents the irregular 6x6 grid network topology used by Boyan and Littman [25] in their experiments. The network consists of two well connected components with a bottleneck of traffic falling on two bridging links. The resulting dependence of network performance on the load is depicted in figure 4.2(top). All graphs represent performance after the policy has converged, averaged over five runs. We tested the network on loads ranging from .5 to 3.5, to compare with the results obtained by Littman and Boyan. The load corresponds to the average number of packets injected per time unit. This is Poisson arrival process and the time till injection of the next packet is distributed exponentially. On this network topology, GAPS is slightly inferior to other algorithms on lower loads, but does at least as well as Bestload on higher loads, outperforming both Q-routing and Best. The slightly inferior performance on low loads is due to exploratory behaviour of GAPS—some fraction of packets is always sent down random link.

To illustrate the difference between the algorithms more explicitly, we altered the network by moving just one link from connecting nodes 32 and 33, to connecting nodes 20 and 27 as illustrated by figure 4.1.right. Since node 20 obviously represents a bottleneck in this configuration, the optimal routing policy is bound to be stochastic. The resulting dependence of network performance on the load is presented in figure 4.2(bottom). GAPS is clearly superior to other algorithms at high loads. It even outperforms "Bestload" that has all the global information in choosing a policy, but is bound to deterministic policies. Notice how the deterministic algorithms get frustrated at much lower loads in this network configuration than in the previous one since from



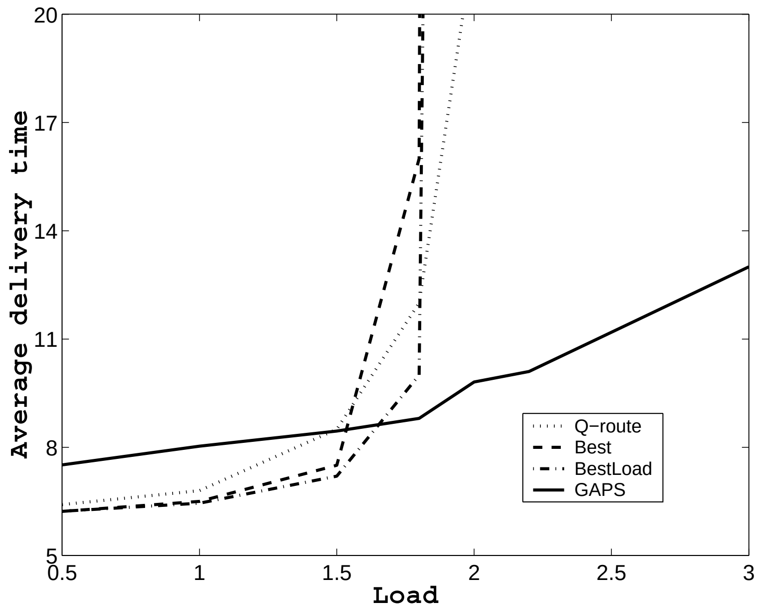


Fig. 4.2: Performance of adaptive routing algorithms on the irregular grid network.

their perspective, the bridge between highly connected components gets twice thinner (compare top and bottom of Figure 4.2).

The GAPS algorithm successfully adapts to changes in the network configuration. Under increased load, the preferred route from the left part of the network to the right becomes evenly split between the two "bridges" at node 20. By using link $20 - 27$, the algorithm has to pay a penalty of making a few extra hops compared to link $20 - 21$, but as the size of the queue at node 21 grows, this penalty becomes negligible compared to the waiting time. Exploratory behavior helps GAPS discover when links go down and adjust the policy accordingly. We have experimented with giving each router a few bits of memory in finite state controller [121, 104] but found that this does not improve performance and slows down the learning somewhat.

4.4 Related Work

The application of machine learning techniques to the domain of telecommunications is a rapidly growing area. The bulk of problems fit into the category of resource allocation; *e.g.*, bandwidth allocation, network routing, call admission control (CAC) and power management. RL appears promising in attacking all of these problems, separately or simultaneously.

While we focus exclusively on routing, Marbach, Mihatsch and Tsitsiklis [93, 94] have applied an actor-critic (value-search) algorithm to tackle both routing and call admission control for calls that differ in value and resource requirements. Their approach is decompositional, representing the network as consisting of link processes, each with its own differential reward. The empirical results on relatively small networks of 4 and 16 nodes show that learning algorithm finds sophisticated policies which are difficult to obtain through heuristics and outperforms these with regard to relative lost reward by 25% to 70%.

Others have addressed admission control as a separate control problem. Carlström [30] introduces predictive gain scheduling, a RL strategy based on a time-series prediction of near-future call arrival rates and precomputation of control policies for Poisson call arrival processes. This approach results in faster learning without performance loss. The online convergence rate increases 50 times on a simulated link with capacity 24 units/sec.

Generally speaking, value-search algorithms have been more extensively investigated than policy search in the domain of communications. Value-search (Q-learning) algorithms have arrived at promising results. Boyan and Littman's [25] algorithm, Q-routing, proves superior to non-adaptive techniques based on shortest paths, and robust with respect to dynamic variations in the simulation on a variety of network topologies, including an irregular 6×6 grid and 116-node LATA phone network. It regulates the trade-off between the number of nodes a packet has to traverse and the possibility of congestion.

Wolpert, Tumer and Frank [190] construct a formalism for the so-called Collective Intelligence (COIN) neural net applied to Internet traffic routing. Their approach involves automatically initializing and updating the local utility functions of individual RL agents (nodes) from the global utility and observed local dynamics. Their simulation outperforms a Full Knowledge Shortest Path Algorithm on a sample network of seven nodes. COIN networks employ a method similar in spirit to the research presented here. They rely on a distributed RL algorithm that converges on local optima without endowing each agent node with explicit knowledge of network topology. However, COIN differs form our approach in requiring the introduction of preliminary structure into the network by dividing it into semi-autonomous neighborhoods that share a local utility function and encourage cooperation. In contrast, all the nodes in our network update their algorithms directly from the global reward.

The work presented in this paper focuses on packet routing using policy search. It resembles the work of Tao, Baxter and Weaver [172], who apply a policy-gradient algorithm to induce cooperation among the nodes of a packet switched network in order to minimize the average packet delay. While their algorithm performs well in several network types, it takes many (tens of millions) trials to converge on a network of just a few nodes. The difference between their results and ours lies in the credit assignment procedure. In our setup, the network is homogeneous, with each link incurring the same cost as any other. In contrast, their experiments reflect a more complex relationship beween network structure and reward.

Applying reinforcement learning to communication often involves optimizing performance with respect to multiple criteria. For a recent discussion on this challenging issue see Shelton [151]. In the context of wireless communication it was addressed by Brown [27] who considers the problem of finding a power management policy that simultaneously maximizes the revenue earned by providing communication while minimizing battery usage. The problem is defined as a stochastic shortest path with discounted infinite horizon, where the discount factor varies to model power loss. This approach resulted in significant (50%) improvement in power usage.

One direction of research seeks to optimize routing decisions by efficiently propagating information about the current state of the network obtained by the traveling packets.Gelenbe et al. [49] divide packets into three types: "smart", "dumb" and "acknowledgment". A small number of smart packets learn the most efficient ways of navigating through the network, dumb packets simply follow the route taken by the smart packets, while acknowledgment packets travel on the inverse route of smart packets to provide source routing information to dumb packets. The division between smart and dumb packets is an explicit representation of the explore/exploit dilemma. Smart packets allow the network to adapt to structural changes while the dumb packets exploit the relative stability between those changes. Promising results are obtained both on a simulation network of 100 nodes and on a physical network of 6 computers.

Subramanian, Druschel and Chen [165] adopt an approach from ant colonies that is very similar in spirit. The individual hosts in their network keep routing tables with the associated costs of sending a packet to other hosts (such as which routers it has to traverse and how expensive they are). These tables are periodically updated by "ants"-messages whose function is to assess the cost of traversing links between hosts. The ants are directed probabilistically along available paths. They inform the hosts along the way of the costs associated with their travel. The hosts use this information to alter their routing tables according to an update rule. There are two types of ants. Regular ants use the routing tables of the hosts to alter the probability of being directed along a certain path. After a number of trials, all regular ants bound to the same destination start using the same routes. Their function is to allow the host tables to converge on the correct cost figure in case the network is stable. Uniform ants take any path with equal probability. They are the ones who continue exploring the network and assure successful adaptation to changes in link status or link cost.

4.5 Discussion

Admittedly, the simulation of the network routing process presented here is far from being realistic. A more realistic model could include such factors as non-homogeneous networks with regard to link bandwidth and routing-node buffer size limits, collisions of packets, packet ordering constraints, various costs associated with say, particular links chosen from commercial versus government subnetworks, and minimal Quality of Service requirements. Introducing priorities for individual packets brings up yet another set of optimization issues. However, the learning algorithm we applied shows promise in handling adaptive telecommunication protocols and there are several obvious ways to develop this research. Incorporating domain knowledge into controller structure is one such direction. It would involve classifying nodes into sub-networks and routing packets in a hierarchical fashion. One step further down this line is employing learning algorithms for routing in ad-hoc networks. Ad-hoc networks are networks where nodes are being dynamically introduced and terminated from the system, as well as existing active nodes are moving about, losing some connections and establishing new ones. Under the realistic assumption that physical variations in the network are slower than traffic routing and evolution, adaptive routing protocols should definitely outperform any heuristic pre-defined routines. We are currently pursuing this line of research.

Chapter 5

Policy Evaluation with Data Reuse

Summary Stochastic optimization algorithms used in reinforcement learning rely on estimates of the value of a policy. Typically, the value of a policy is estimated from results of simulating that very policy in the environment. This approach requires a large amount of simulation as different points in the policy space are considered. In this chapter, we develop value estimators that use data gathered when using one policy to estimate the value of using another policy, resulting in much more data-efficient algorithms.

5.1 Introduction

So far, we have focused on designing algorithms for an agent interacting with an environment, to adjust its behavior in such a way as to optimize a long-term return. This means searching for an optimal behavior in a class of behaviors. The success of learning algorithms therefore depends both on the richness of information about various behaviors and on how effectively it is used. While the latter aspect has been given a lot of attention, the former aspect has not been addressed carefully. In this chapter we adapt solutions developed for similar problems in the field of statistical learning theory.

The process of interaction between the learning agent and the environment is costly in terms of time, money or both. Therefore, it is important to carefully allocate available interactions, to use all available information efficiently and to have an estimate of how informative the experience overall is with respect to the class of possible behaviors.

In realistic settings, the class of policies is restricted and even among the restricted set of policies, the absolute best policy is not expected to be found due to the difficulty of solving a global multi-variate optimization problem. Rather, the only option is to explore different approaches to finding near-optimal solutions among local optima.

The issue of finding a near-optimal policy from a given class of policies in reinforcement learning is analogous to a similar issue in *supervised learning*. There we are looking for a near-optimal hypothesis from a given class of hypotheses [179]. However, there are crucial differences in these two settings. In supervised learning we assume that there is some target function that labels the examples, generated according to some distribution. This distribution is the same for all the hypotheses. It implies both that the same set of samples can be evaluated on any hypothesis, and that the observed error is a good estimate of the true error.

In contrast, there is no fixed distribution generating experiences in reinforcement learning. Each policy induces a different distribution over experiences. The choice of a policy defines both a "hypothesis" and a distribution. This raises the question of how one can reuse the experience obtained while following one policy to learn about another. The other policy might generate a very different set of samples (experiences), and in the extreme case the support of the two distributions might be disjoint.

We present a way of reusing all of the accumulated experience by an agent that does not have a (generative) model of the environment (see [73]). We make use of a technique known as "importance sampling" [138] or "likelihood ratio estimation" [51] to different communities in order to estimate the policy's value. We also discuss properties of various estimators. The general idea is illustrated by Figure 5.1. Our goal is to build a module that contains a non-parametric model of the optimization surface. Given an arbitrary policy, such a module outputs an estimate of its value, as if the policy had been tried out in the environment. With such a module, guaranteed to provide good estimates of policy value across the policy class, we may use our favorite optimization algorithm.

One realistic off-line scenario in reinforcement learning is when

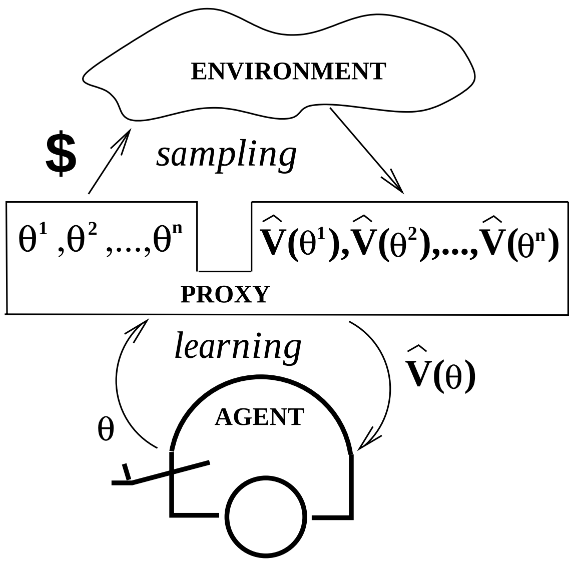

ENVIRONMENT
$
sampling
$\theta^1, \theta^2, \ldots, \theta^n$
$\hat{V}(\theta^1), \hat{V}(\theta^2), \ldots, \hat{V}(\theta^n)$
PROXY
learning
$\hat{V}(\theta)$
θ
AGENT

Fig. 5.1: An organization of the policy evaluation process.

the data processing and optimization (learning) module is separated (physically) from the data acquisition module (agent). Say we have an ultra-light micro-sensor connected to a central computer. The agent then has to be instructed initially how to behave when given a chance to interact with the environment for a limited number of times, then bring/transmit the collected data back. Naturally, during such limited interaction only a few possible behaviors can be tried out. It is extremely important to be able to generalize from this experience in order to make a judgment about the quality of behaviors that were not tried out. This is possible when some kind of similarity measure in the policy class can be established. If the difference between the values of two policies can be estimated, we can estimate the value of one policy based on experience with the other.

The rest of this chapter is organized as follows. Section 5.2 presents the necessary background in sampling theory and presents our way of estimating the value of policies. The algorithm is described in Section 5.3. Section 5.4 presents comparative empirical studies of likelihood sampling combined with gradient descent in various implementations.

5.2 Likelihood Ratio Estimation

For the sake of clarity, we will review concepts from sampling theory using relevant reinforcement learning concepts. Rubinstein [138] provides a good overview of this material. Note that for simplicity we use h_i to denote experience as both the random variable and its realisation. We believe this does not introduce any confusion into the presentation.

"Crude" sampling If we need to estimate the value $V(\theta)$ of policy θ, from independent, identically distributed (i.i.d.) samples induced by this policy, after taking N samples $h_i, i \in (1..N)$ we have:

$$\hat{V}(\theta) = \frac{1}{N}\sum_i R(h_i) \ . \tag{5.1}$$

The expected value of this estimator is $V(\theta)$ and it has variance:

$$\begin{aligned}\mathrm{Var}\left[\hat{V}(\theta)\right] =& \frac{1}{N}\sum_{h\in H} R(h)^2 \Pr(h|\theta) - \frac{1}{N}\left[\sum_{h\in H} R(h)\Pr(h|\theta)\right]^2 \\ =& \frac{1}{N}\mathrm{E}_\theta\left[R(h)^2\right] - \frac{1}{N}V^2(\theta) \ .\end{aligned}$$

Indirect sampling Imagine now that for some reason we are unable to sample from the policy θ directly, but instead we can sample from another policy θ'. The intuition is that if we knew how "similar" those two policies were to one another, we could use samples drawn according to the distribution $\Pr(h|\theta')$ and make an adjustment according to the similarity of the policies. Formally we have:

$$\begin{aligned}V(\theta) &= \sum_{h\in H} R(h)\Pr(h|\theta) = \sum_{h\in H} R(h)\frac{\Pr(h|\theta')}{\Pr(h|\theta')}\Pr(h|\theta) \\ &= \sum_{h\in H} R(h)\frac{\Pr(h|\theta)}{\Pr(h|\theta')}\Pr(h|\theta') = \mathrm{E}_{\theta'}\left[R(h)\frac{\Pr(h|\theta)}{\Pr(h|\theta')}\right] \ .\end{aligned}$$

Note that the agent is not assumed to know the environment's dynamics, which means that it might not be (and most often is not) able to calculate $\Pr(h|\theta)$. However, we can prove the following key result.

Theorem 5.1 *For any* POMDP *and any policy space* Θ*, such that the probability of having any experience* h *under any policy* $\theta \in \Theta$

is bounded away from zero, it is possible for an agent to calculate the ratio $\frac{\Pr(h|\theta)}{\Pr(h|\theta')}$ *for any* $\theta, \theta' \in \Theta$ *and* $h \in H$.

Proof: The Markov assumption in POMDPs warrants that $\Pr\left(h|\theta\right)$

$$= \Pr\left(s(0)\right) \prod_{t=1}^{T} \Pr\left(o(t)|s(t)\right) \Pr\left(a(t)|o(t), \theta\right) \Pr\left(s(t{+}1)|s(t), a(t)\right)$$
$$= \left[\Pr(s(0)) \prod_{t=1}^{T} \Pr\left(o(t)|s(t)\right) \Pr\left(s(t{+}1)\middle|s(t), a(t)\right)\right] \left[\prod_{t=1}^{T} \Pr(a(t)|o(t), \theta)\right]$$
$$= \Psi\big(h\big) \Phi\big(\theta, h\big)\ .$$

$\Psi(h)$ is the factor in the probability related to the part of the experience, dependent on the environment, that is unknown to the agent and can only be sampled. $\Phi(\theta, h)$ is the factor in the probability related to the part of the experience, dependent on the agent, that is known to the agent and can be computed (and differentiated). Note that $\Pr\left(h|\theta\right)$ can be broken up this way both when the controller executes a reactive policy and when it executes a policy with internal state (see for example the derivation for internal state sequences in Shelton's dissertation [151]). Therefore we can compute

$$\frac{\Pr(h|\theta)}{\Pr(h|\theta')} = \frac{\Psi(h)\Phi(\theta, h)}{\Psi(h)\Phi(\theta', h)} = \frac{\Phi(\theta, h)}{\Phi(\theta', h)}\ .$$

□

We can now construct an indirect estimator $\hat{V}^{IS}_{\theta'}(\theta)$ of $V(\theta)$ from i.i.d. samples $h_i, i \in (1..N)$ drawn according to the distribution $\Pr(h|\theta')$:

$$\hat{V}^{IS}_{\theta'}(\theta) = \frac{1}{N} \sum_{i} R(h_i) w_\theta(h_i, \theta')\ \ , \tag{5.2}$$

where for convenience, we denote the fraction $\frac{\Pr(h|\theta)}{\Pr(h|\theta')}$ by $w_\theta(h, \theta')$. This is an unbiased estimator of $V(\theta)$ with variance

$$\begin{aligned} \operatorname{Var}\left[\hat{V}^{IS}_{\theta'}(\theta)\right] =& \frac{1}{N} \left\{ \sum_{h \in H} \left(R(h) w_\theta(h, \theta')\right)^2 \Pr(h|\theta') - V(\theta)^2 \right\} \\ =& \frac{1}{N} \left\{ \sum_{h \in H} \frac{\left(R(h) \Pr(h|\theta)\right)^2}{\Pr(h|\theta')} - V(\theta)^2 \right\} \\ =& \frac{1}{N} E_\theta \left[R(h)^2 w_\theta(h, \theta')\right] - \frac{1}{N} V(\theta)^2\ \ . \end{aligned} \tag{5.3}$$

This estimator $\hat{V}^{\mathrm{IS}}_{\theta'}(\theta)$ is usually called in statistics [138] an *importance sampling* (IS) estimator because the probability $\Pr(h|\theta')$ is chosen to emphasize parts of the sampled space that are important in estimating V. The technique of IS was originally designed to increase the accuracy of Monte Carlo estimates by reducing their variance [138]. Variance reduction is always a result of exploiting some knowledge about the estimated quantity.

Optimal sampling policy It can be shown [69], for example by optimizing expression 5.3 with Lagrange multipliers, that the optimal sampling distribution is $\Pr(h|\theta') = \frac{R(h)\Pr(h|\theta)}{V(\theta)}$, which gives an estimator with *zero* variance. Not surprisingly this distribution cannot be used, since it depends on prior knowledge of a model of the environment (transition probabilities, reward function), which contradicts our assumptions, and on the value of the policy, which is what we need to calculate. However all is not lost. There are techniques that approximate the optimal distribution, by changing the sampling distribution during the trial, while keeping the resulting estimates unbiased via reweighting of samples, called "adaptive importance sampling" and "effective importance sampling" (see, for example, [113, 192, 117]). In the absence of any information about $R(h)$ or evaluated policy, the optimal sampling policy is the one that selects actions uniformly at random.

On average, the weighting coefficient $w_\theta(h, \theta')$ is equal to one, but unfortunately the variance of this quantity can get arbitrary large. Formally,

Lemma 5.1 $E_{\theta'}[w_\theta(h, \theta')] = 1.$

Proof:

$$E_{\theta'}[w_\theta(h, \theta')] = \sum_h \frac{\Pr(h|\theta)}{\Pr(h|\theta')}\Pr(h|\theta') = \sum_h \Pr(h|\theta) = 1.$$

□

Lemma 5.2 $\sigma_{\theta'}[w_\theta(h, \theta')] = E_\theta[w_\theta(h, \theta')] - 1.$

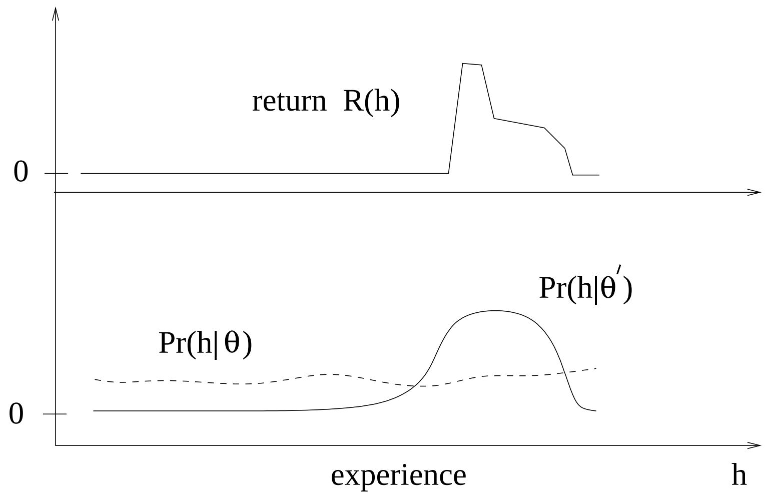


Fig. 5.2: Hypothetical return and probability functions.

Proof:

$$\begin{aligned}\mathrm{Var}_{\theta'}[w_\theta(h,\theta')] =& \mathrm{E}^2_{\theta'}[w_\theta(h,\theta')] - \mathrm{E}_{\theta'}[(w_\theta(h,\theta'))^2] \\ =& \mathrm{E}^2_{\theta'}[w_\theta(h,\theta')] - 1 \\ =& \sum_h (w_\theta(h,\theta'))^2 \Pr(h|\theta') - 1 \\ =& \sum_h \left(\frac{\Pr(h|\theta)}{\Pr(h|\theta')}\right)^2 \Pr(h|\theta') - 1 \\ =& \sum_h \left(\frac{\Pr(h|\theta)}{\Pr(h|\theta')}\right) \Pr(h|\theta) - 1 \\ =& \mathrm{E}_\theta[w_\theta(h,\theta')] - 1 \;.\end{aligned} \tag{5.4}$$

□

Intuitively, the higher $\Pr(h|\theta)$, the higher is $w_\theta(h,\theta')$ and the more likely it is to be encountered.

To get a further intuition we will need two definitions:

Definition 1 *A* KL-distance *between two distributions is defined as*

$$D_{KL}(p,q) = \sum_x \ln\left(\frac{p(x)}{q(x)}\right) q(x);$$

Definition 2 (Jensen's inequality) *For any convex function* $f(x)$, $E[f(x)] \geq f(E[x])$.

We can establish:

$$\begin{aligned} E_\theta[w_\theta(h,\theta')] =& E_\theta \exp\left(-\log\frac{\Pr(h|\theta')}{\Pr(h|\theta)}\right) \\ &\geq \exp\left(-D_{KL}\left(\Pr(h|\theta'),\Pr(h|\theta)\right)\right) . \end{aligned}$$

Remark **5.2.1** *It is sometimes possible to get a better estimate of* $V(\theta)$ *by following another policy* θ'*, rather than the policy* θ *itself*[1]*. Here is an illustrative example (see figure 5.2): imagine that a return function* $R(h)$ *is such that it is* zero *for all experiences in some sub-space* H_0 *of experience space* H*. At the same time the policy* θ*, which we are trying to evaluate spends almost all the time there, in* H_0*. If we follow* θ *in our evaluation, we are wasting samples and time! It makes sense to use another policy, which induces experiences with non-zero return, and reweight the samples*[2]*.*

Weighted indirect sampling Powell and Swann [130] introduced *weighted uniform sampling* (also described by Rubinstein [138]). It achieves a variance reduction by using the distribution $\Pr(h|\theta')$ for reweighting, while drawing samples from the original distribution $\Pr(h|\theta)$:

$$\hat{V}_{\theta'}(\theta) = \frac{\sum_i R(h_i)\Pr(h|\theta)}{\sum_i \Pr(h|\theta')} . \tag{5.5}$$

Estimator (5.5) is good for cases when it is easier to draw samples according to estimated θ than according to some other θ', which however reflects some knowledge about the returns.

The weighted uniform estimator was extended by Spanier [163] to random walk processes and generalized to

$$\hat{V}_{\theta'}(\theta) = \frac{\sum_i R(h_i)\frac{\Pr(h|\theta)}{\Pr(h|\theta^1)}}{\sum_i \frac{\Pr(h|\theta^2)}{\Pr(h|\theta^1)}} . \tag{5.6}$$

[1]This remark is due to Luis Ortiz [116] whose advise we would like to acknowledge.

[2]In this case, we can really call what happens *"importance sampling"*, unlike usually when it is just "reweighting", not connected to "importance" *per se*. That is why we advocate using the name *"likelihood ratio"* rather than *"importance sampling"*.

where we sample according to θ^1, which reduces to estimator (5.5) upon having uniform $\Pr(h|\theta^1)$, to estimator (5.2) upon having $\theta^1 \equiv \theta^2$, and to estimator (5.1) upon having both $\Pr(h|\theta^1)$ and $\Pr(h|\theta^2)$ uniform.

The principal properties of the estimator (5.6) were established by Spanier [163]. This is a biased but consistent estimator of $V(\theta)$, meaning that its bias β tends to zero as number of samples increases. Furthermore, under relatively mild assumptions, the mean square error (MSE) of this estimator, which is the sum of the variance and the bias squared ($\text{MSE} = \text{Var} + \beta^2$) is

$$\begin{aligned}\text{MSE}\left[\hat{V}_{\theta'}(\theta)\right] &= \frac{1}{N}\left\{\sum_{h\in H}\Big(\big(R(h) - V(\theta)\big)w_\theta(h,\theta')\Big)^2 \Pr(h|\theta')\right\} + O\left(\frac{1}{N}\right) \\ &= \frac{1}{N}\left\{\sum_{h\in H}\frac{\big(R(h)\Pr(h|\theta)\big)^2}{\Pr(h|\theta')} - V(\theta)^2\right\} + O\left(\frac{1}{N}\right). \end{aligned} \tag{5.7}$$

Finally, it turns out that a particular choice of $\theta^2 \equiv \theta$ has a remarkably small mean square error over a large range of problems encountered in the field of particle transform modeling [163]. This estimator was also used in the recent work of Sutton and Precup [131] in the context of exploration for the TD(λ) family of on-line learning algorithms. Let us call this estimator the *weighted importance sampling* (WIS) estimator.

$$\hat{V}^{\text{WIS}}_{\theta'}(\theta) = \frac{\sum_i R(h_i)w_\theta(h_i,\theta')}{\sum_i w_\theta(h_i,\theta')} \, . \tag{5.8}$$

Remark **5.2.2** *Let us try to understand the difference between $\hat{V}^{\text{IS}}_{\theta'}(\theta)$ and $\hat{V}^{\text{WIS}}_{\theta'}(\theta)$ estimators. Intuitively, the variance is high for the sampling policy θ' if there are some trajectories with extremely low probabilities under θ', which have relatively significant probabilities under the estimated policy θ, because it causes the value $w_\theta(h,\theta')$ to blow up. The* WIS *estimator compensates for that by normalizing by the sum of the weights.*

Remark **5.2.3** *Let's look at some assymptotics of the two estimators. When the number of samples used is huge, bias of* WIS *tends to zero $\left(\frac{\beta^2}{\text{MSE}} = o\left(\frac{1}{N}\right)\right)$, so it is really on estimating with just a few samples, when behavior is different.* IS *has a larger* *variance, while* WIS *has a smaller variance offset by bias. That means that we can have more certainty about the outcome of the* WIS *estimator. In the extreme, if we were to use only one sample, the* WIS *gives us* $\hat{V}^{WIS}_{\theta',1}(\theta) = \hat{V}^{WIS}_{\theta',1}(\theta')$ *with* $E_{\theta'}\left[\hat{V}^{WIS}_{\theta',1}(\theta)\right] = V(\theta')$ *and a variance* $Var_{\theta'}\left[\hat{V}^{WIS}_{\theta',1}(\theta)\right] = E_{\theta'}\left[R(h)^2\right] - V(\theta')^2$, *which is a quite well-behaved variance. For* IS $Var_{\theta'}\left[\hat{V}^{IS}_{\theta',1}(\theta)\right] = E_{\theta'}\left[R(h)^2\left(\frac{Pr(h|\theta)}{Pr(h|\theta')}\right)^2\right] - V(\theta)^2$, *which has the apparent danger of having a probability in the denominator.*

Multiple sampling policies So far, we talked about using a single policy to collect all samples for estimation. We also made an assumption that all considered distributions have equal support. In other words, we assumed that any history has a non-zero probability to be induced by any policy. Obviously it could be beneficial to execute a few different sampling policies, which might have disjoint or overlapping support. There is literature on this so-called stratification sampling technique [138]. Here we just mention that it is possible to extend our analysis by introducing a prior probability on choosing a policy out of a set of sampling policies, then executing this sampling policy. Our sampling probability will become: $Pr(h) = Pr(\theta')\,Pr(h|\theta')$.

5.3 Policy Evaluation Algorithm

We have discussed various ways of evaluating a policy by combining samples of the return function. In this section we describe how to turn this estimation process into learning algorithm.

As illustrated by Figure 5.1, we wish to build a *proxy environment* that contains a non-parametric model of the values of all policies. Given an arbitrary policy, the proxy environment returns an estimate of its value, as if the policy were tried out in the real environment. We assume that obtaining a sample from the environment is costly, so we want the proxy module to make only a small number of queries regarding policies $\{\theta^i\}, i = 1..N$ and receive estimates $\{\hat{V}(\theta^i)\}$. These queries are implemented by the `sample()` routine (Table 5.1). It requires memory of size[3] $O(n|S||A|)$ and does not depend on the length of the horizon

[3]We have n records for trajectories, each of which contains: one number R^i, one number Φ^i, $|S|$ counters N_s of visiting states and $|S||A|$ counters N_{sa} of performing

Tab. 5.1: The sample() routine.

```
**Input:** set of policies $\theta^i$, $i \in 1..n$
**For** $i = 1$ to $n$: (record $n$ trials)
    • Beginning of the trial:
        $\Phi^i \leftarrow 1$, $R \leftarrow 0$
        for all $(o, a)$:
            $N_o \leftarrow 0$, $N_{oa} \leftarrow 0$
        initialize the state $s(0)$, observe $o(0)$
    • At each time step $t$ of the trial:
        inc $\left(N^i_{o(t)}\right)$
        Draw next $a(t)$ from $\mu(o(t), a(t), \theta^i)$
        inc $\left(N^i_{o(t)a(t)}\right)$
        Execute $a(t)$, moving environment into $s(t)$
        Get reward $r(t)$, new observation $o(t)$
        $R^i \leftarrow R^i + \gamma^t r(t)$
        $\Phi^i \leftarrow \Phi^i \mu\left(o(t), a(t), \theta^i\right)$
**Output:** $D = \left\langle N^i_o, N^i_{oa}, R^i, \Phi^i \right\rangle$
```

T, since all we keep are counters of events in the trajectory. We denote by Φ^i, $i \in 1..n$, numbers calculated in procedure sample() according to $\prod_{t=1}^{T} \Pr(a(t)|o(t), \theta^i)$.

Any policy search algorithm can now be combined with this proxy environment to learn from scarce experience. As an example we provide the optimize() routine (Table 5.2), which calculates an estimate of the gradient and performs a steepest descent update $\theta_{oa} \leftarrow \theta_{oa} + \alpha \Delta_{oa}$ when the policy is represented by a look-up table (see page 24) and the Boltzmann law is used (see equation (2.3) in section 2.2). It is possible to substitute for this update line a call to line-search [132], which relies on value estimates and does not require a learning rate. The optimize() routine computes an IS estimate of the gradient from

an action a in state s. In case of non-stationary policies we also have to keep nT numbers for n trajectories of horizon T. Rigorously speaking it has a logarithmic dependence on the length of the horizon.

Tab. 5.2: The optimize() routine.

```
Input: initial policy $\theta$, $D = \langle N_o^i, N_{oa}^i, R^i, \Phi^i \rangle$,
       # of optimization steps $m$
For $j = 1$ to $m$: (do $m$ optimization steps)
    Init: For all $(o, a)$: $\Delta_{oa} \leftarrow 0$, $\kappa \leftarrow 0$
    For $i = 1$ to $n$: (index recorded trials)
        Initialize: $\Phi(\theta) \leftarrow 1$
        For all $(o, a)$:
            $\Phi(\theta) \leftarrow \Phi(\theta)\mu(o, a, \theta)^{N_{oa}^i}$
        For all $(o, a)$
            $\Delta_{oa} \leftarrow \Delta_{oa} + R^i \frac{\Phi(\theta)}{\Phi^i} \left(N_o^i - \mu(o, a, \theta^i) N_{oa}^i\right)$
        $\kappa \leftarrow \kappa + \Phi(\theta)/\Phi^i$
    For all $(o, a)$:
        $\theta_{oa} \leftarrow \theta_{oa} + \alpha \Delta_{oa} \frac{1}{\kappa}$
Output: hypothetical optimum $\theta$
```

values

$$\Phi^i = \prod_{t=1}^{T} \mu(o(t), a(t), \theta) = \prod_{o \in O, a \in A} \mu(o, a, \theta)^{N_{oa}^i} \ .$$

To compute a WIS estimate we would do

$$\Delta_{oa}^{\text{WIS}} = \frac{1}{\sum_{i=1}^{n} \frac{\Phi(\theta)}{\Phi^i}} \sum_{i=1}^{n} R^i \frac{\Phi(\theta)}{\Phi^i} \left(N_o^i - \mu(o, a, \theta^i) N_{oa}^i\right) \ .$$

Remark **5.3.1** *Note that it is possible to decide which estimator to use a posteriori, based on the estimation of bias and variance.*

How to sample? Different approaches can be taken to learning, corresponding to different combinations of the sample and optimize routines. One is to postpone optimization until all interaction with the environment is completed, and combine all the available information in order to estimate (*off-line*) the whole "value surface" over the policy parameter space. Formally it is presented in Table 5.3.a. Another

Tab. 5.3: Various ways to combine sample() with optimize().

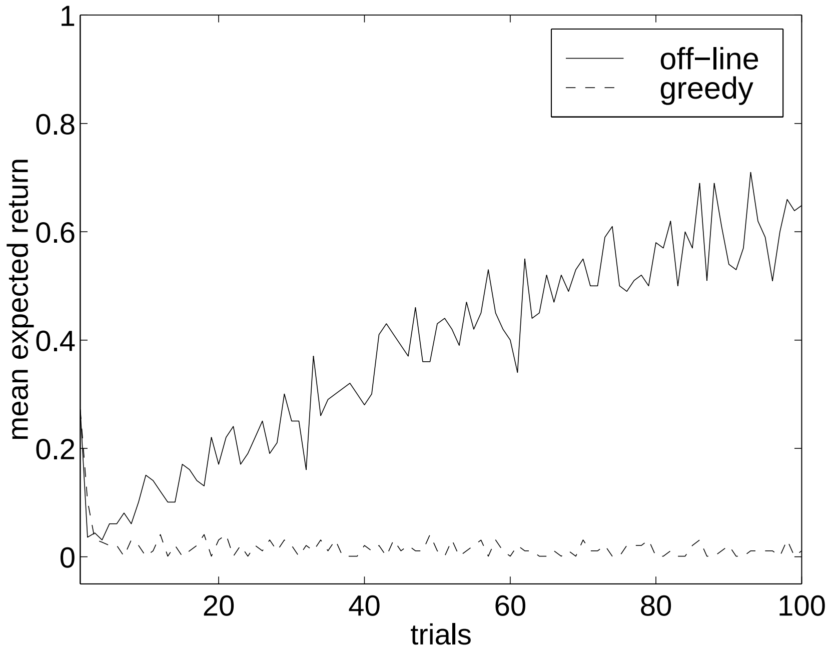
```
$D \leftarrow 0$, $\theta \leftarrow u$;

$D \leftarrow$ sample($\theta, n$);
$\theta^* \leftarrow$ optimize($\theta, D, \infty$);

a. "Off-line"
```

```
$\theta \leftarrow \theta_0$, $D \leftarrow 0$;
Loop:
  $D \leftarrow$ sample($\theta, n$);
  $\theta \leftarrow$ optimize($\theta, D, m$);

b. "Forget"
```

```
$D \leftarrow 0$, $\theta \leftarrow \theta_0$;
Loop:
  $D \leftarrow D \cup$ sample($\theta, n$);
  $\theta \leftarrow$ optimize($\theta, D, m$);

c. "Greedy"
```

approach involves using an algorithm, driven by newly generated policy value (and gradient thereof) estimates at each iteration, to update the hypothesis about the optimal policy after each interaction (or few interactions) with the environment. We will call this *on-line* or learning with greedy sampling. Table 5.3.c presents this brute-force way to do on-line optimization while reusing all experience at every point of time. The obvious drawback is of course that the size of dataset D grows infinitely with time. An alternative is to use the data from a few runs to get an unbiased estimate of the gradient, use it to make a (few) step(s) up the gradient, then forget it (table 5.3.b). The REINFORCE algorithms [104] are an extreme case of "forget", in which each trial is followed by one optimization step ($n = m = 1$).

Illustration: Bandit Problems Let us consider a trivial example of a bandit problem to illustrate the algorithms from the previous section. The environment has a degenerate state space of one state, in which two actions a_1 and a_2 are available. The space of policies available is stochastic and encoded with one parameter p, the probability



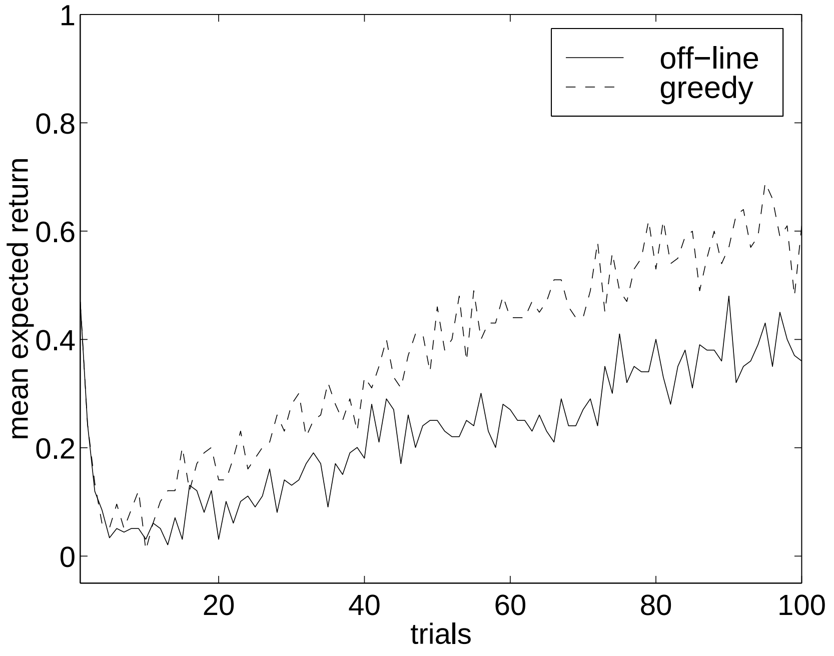


Fig. 5.3: Empirical results for "hidden treasure" (top) and "hidden failure" (bottom) problems.

of taking the first action, which is constrained to be in the interval $[\, \underline{c}, \overline{c}\,] = [.2, .8]$. We consider two problems, called "HT" (hidden treasure) and "HF" (hidden failure) both of which have the same expected returns for actions: 1 for a_1 and 0 for a_2. In both HF and HT a_1 always returns 1. In HF a_2 returns 10 with probability .99 and -990 with probability .01. In HT, a_2 always returns 0, while a_1 returns -10 with probability .99 and $+1090$ with probability .01. We would expect a *greedy* learning algorithm to sample near policies that look better under scarce information, tending to choose the sub-optimal a_2 in the HT problem. This strategy is inferior to *off-line* (or blind) sampling, which samples uniformly from the policy space and will discover the hidden treasure of a_1 faster. This is indeed the behavior we observe in simulations (see figure 5.3.left)[4]. On the contrary, for the HF problem (Figure 5.3.right) the greedy algorithm does better, by initially concentrating on a_2, which looks better, but discovering the hidden failure quite soon. In this problem, the off-line sampling algorithm is slower to discover the failure. Note that although greedy is somewhat better in HF, it is much worse in HT. It illustrates why, without any prior knowledge of the domain and given a limited number of samples, it is important not to guide sampling solely by optimization.

5.4 Empirical Studies of Likelihood Ratios for Exploration

In this section, we illustrate on a simple domain how importance sampling may be used in GAPS to enable efficient exploration and thereby dramatically speed up learning. We compare the performance of three algorithms. The naive algorithm described in section 2 which we call *on-policy*, and two versions of the *off-policy* algorithm described in this chapter, using both IS and WIS estimators, which we address as IS and WIS algorithms correspondingly.

Since all we need to evaluate some policy θ from results of following another policy[5] θ' is to calculate the likelihood ratio, the exploratory policy θ' may be arbitrary—it does not have to be stationary as does θ. In particular, θ' can use any type of extra information such as counters of state visits. The only requirement is that any trajectory

[4]All graphs represent an average result over 30 runs.

[5]That is what gives the name *off-policy* to this class of algorithms.

possible under θ is still possible under θ'. Hence we cannot use any *deterministic* policy. But we can, for instance, mix such a policy with the current policy using an exploration ratio λ; *i.e.*, at each time step we follow a deterministic policy with probability $\lambda \in [0, 1)$, and the current policy θ with probability $(1 - \lambda) > 0$. That enables efficient exploration strategies.

The off-policy algorithm we use for these experiments is a "forget" algorithm presented in table 5.3.b with n sampling trials followed by one optimization step. During the trials, counters of visited states and performed actions are maintained and a cumulative likelihood ratio is computed. After n trials an update to the current policy is made according to the gradient estimate computed by the IS or WIS estimator correspondingly. Table 5.4 presents in detail the algorithm obtained if we use WIS estimator, look-up table policy representation and Boltzmann law. This algorithm actually achieved the best performance of all the algorithms we compared.

The counter-based exploration policy gives such an advantage to the *off-policy* algorithm for the following reason. Depending on the way the controller is initialized in the beginning of learning, the complexity of the first trial(s) may be very bad due to the initial random walk of the algorithm. Since the update resulting from a single trial may not change the policy a lot, one may observe a very bad performance during several trials in the beginning of learning. Changing the reward model—as suggested by Koenig and Simmons [76] for QL algorithms in MDP—may not reduce the expected length of GAPS's very first trial since there are no weight updates *during* a trial, and the length of the first trial depends only on the initial controller.

The off-policy implementations of GAPS can be used with efficient directed exploration policies to avoid an initial random walk, with θ' set to many directed exploration policies (which are often deterministic and non stationary), including Thrun's counter-based [174] and Meuleau and Bourgine's global exploration policy [101]. Moreover, the algorithms can easily be adapted to stochastic exploration policies.

Numerical Simulations

We tested our off-policy algorithms on a simple grid-world problem consisting of an empty square room where the starting state and the goal are two opposite corners (see figure 5.4). Four variants of this

Tab. 5.4: An off-policy implementation of GAPS based on weighted IS.

```
1. Initialize the controller weights $\theta_{oa}$
2. Initialize variables:
   • for all $(o, a)$: $\Delta_{oa} \leftarrow 0$
   • $\mathcal{K} \leftarrow 0$
3. For $i = 1$ to $n$:                    (executes $n$ learning trials)
   • Beginning of a trial:
       for all $(o, a)$: $N_o \leftarrow 0$, $N_{oa} \leftarrow 0$
       $R \leftarrow 0$
       $K \leftarrow 1$
       $h \leftarrow (o(0))$
   • At each time-step $t$ of the trial:
       inc $(N_{o(t)})$
       with probability $\lambda$: $a(t) \leftarrow \mu(h, \theta')$
           with prob. $1 - \lambda$: draw $a(t)$ at random following $\mu(o(t), \cdot, \theta)$
       if $a(t) = \mu(h, \theta')$: $K \leftarrow K\mu(o(t), a(t))/(\lambda + (1 - \lambda)\mu(o(t), a(t), \theta))$
           else: $K \leftarrow K/(1 - \lambda)$
       inc $(N_{o(t)a(t)})$
       execute an action $a(t)$, receive $r(t), o(t + 1)$ from environment
       $R \leftarrow R + \gamma^t r(t)$
       append the triple $(a(t), r(t), o(t + 1))$ to $h$
   • End of the trial:
       for all $(o, a)$: $\Delta_{oa} \leftarrow \Delta_{oa} + KR(N_{oa} - \mu(o, a, \theta)N_o)/\zeta$
       $\mathcal{K} \leftarrow \mathcal{K} + K$
4. Update policy:
   • for all $(o, a)$: $\theta_{oa} \leftarrow \theta_{oa} + \alpha\Delta_{oa}/\mathcal{K}$
5. Loop: return to 2.
```

problem were tried: there may or may not be a reset action, and the problem can be fully observable or partially observable. The reset action brings the agent back to the starting state. When the problem is partially observable, the agent cannot perceive its true location, but only the presence or absence of walls in its immediate proximity. This perceptual aliasing is illustrated by shading on figure 5.4. Each square with the same shading looks the same to the agent. This problem was not designed to be hard for the algorithms, and every version of GAPS converges easily to the *global* optimum. However, it allows us to compare closely the different variants in terms of learning speed.

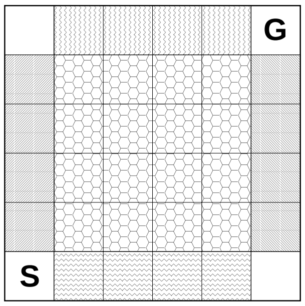


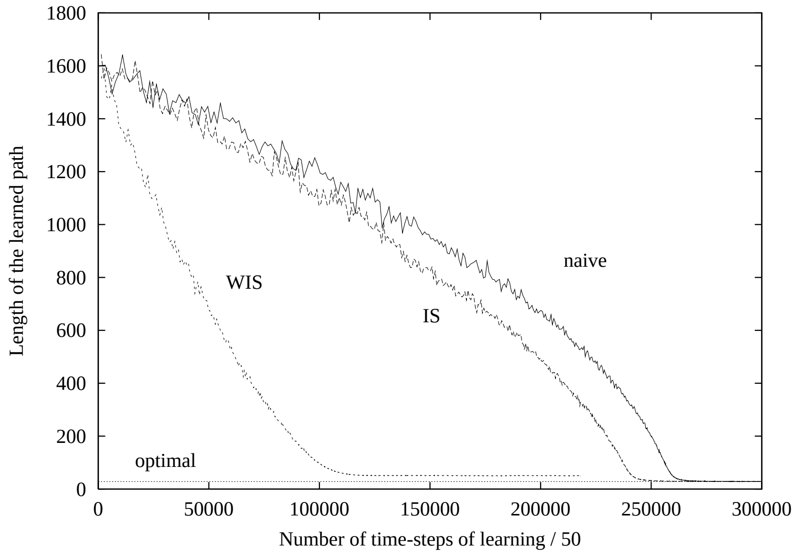
Fig. 5.4: A simple grid-world problem.

Figure 5.5 present learning curves obtained in the 15×15 partially-observable no-reset variant of the problem, using Meuleau and Bourgine's [101] global exploration policy with $\gamma = 0.95$, $\alpha = 0.01$, $n = 3$, $\zeta = 1$, averaged over 30 runs. These graphs represent the evolution of the quality of the policy learned as a function of the total number of time-steps of interactions with the environment.

Controllers were initialized by the uniform distribution on actions for all observations. We tried several parameter settings and environments of different sizes. In the choice of the sampling policy, we focused on the objective of reducing the trial's length: we tried several directed exploration policies as θ', including greedy counter-based, Thrun's counter-based [174] and an indirect (QL) implementation of a global counter-based exploration policy proposed by Meuleau and Bourgine [101]. Exploration strategies designed for fully observable environments were naively adapted by replacing states by observations in the formulae, when dealing with the partially observable variant of the problem.

The results of these experiments are qualitatively independent of the variant and the size of the problem (although we were unable to run experiments in reset problems of reasonable size, due to the exponential complexity of random walk in these problems). The best performance was obtained using Meuleau and Bourgine's global exploration policy [101].

In general, *on-policy* sampling is very stable and slow. With small values of λ, simple IS allows us to reduce the length of learning trials without affecting the quality of policy learned. However the IS algorithm rapidly becomes unstable and systematically jumps to poor policies as the ratio λ increases. Compare figure 5.5(top) which presents results for $\lambda = .1$ with figure 5.5(bottom) which presents results for



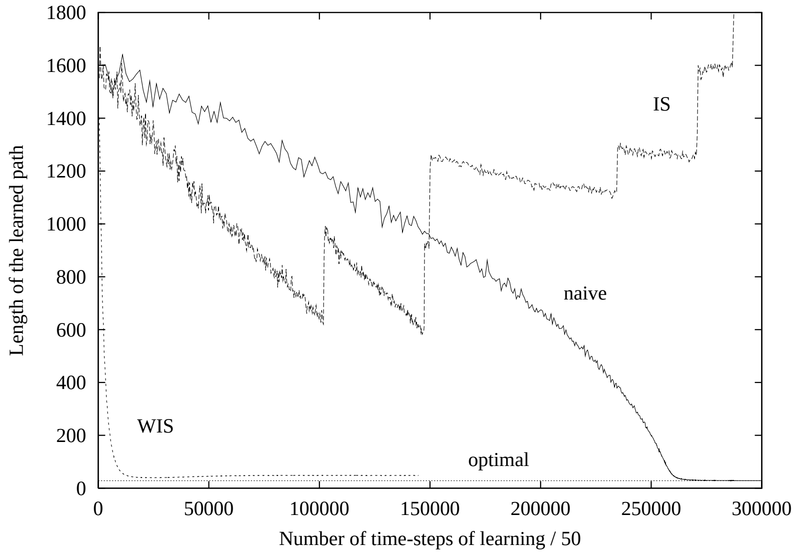


Fig. 5.5: Learning curves for partially observable room (exploration ratios are 0.1 (top) and 0.3 (bottom)).

$\lambda = .3$. This instability is a known drawback of IS when the sampling distribution (the strategy used during learning) differs a lot from the target distribution (the current policy) [193]. In this case, very unlikely events are associated with huge importance coefficients. Hence, whenever they happen, they induce devastating weight updates that can push the algorithm to a very bad policy.

The WIS algorithm is by far the most efficient algorithm. It stays stable when λ approaches 1, even with a relatively small number of learning trials ($n = 5$). It can thus be used with high values of λ, which allows dramatic reduction of the trials' length. This empirical findings are in accordance with theoretical results of section 5.2.

5.5 Discussion

In this chapter, we developed value estimators that use data gathered when using one policy to estimate the value of using another policy, resulting in data-efficient algorithms.

Stochastic gradient methods and likelihood ratios have been long used for optimization problems (see work of Peter Glynn [50, 51, 52, 53] and related fundamental texts [47, 48, 112, 29]). Recently, stochastic gradient descent methods, in particular REINFORCE [189, 186, 187], have been used in conjunction with policy classes constrained in various ways; e.g., with external memory [123], finite state controllers [104] and in multi-agent settings [122]. Furthermore, the idea of using likelihood ratios in reinforcement learning was initially suggested by Szepesvari [169] and developed for solving MDPs with function approximation by Precup *et al.* [131] and for gradient descent in finite state controllers by Meuleau *et al.* [103]. However only on-line optimization was considered. Shelton [152, 151] developed greedy algorithm for combining samples from multiple policies in normalized estimators and demonstrated a dramatic improvement in performance.

In some domains there is a natural distance between observations and actions, which allows us to re-use experience without likelihood ratio estimation. One such domain is financial planing and investments. See the paper by Glynn *et al.* [115] for the description of kernel-based RL algorithm.

There is room for employing various alternative sampling techniques, in order to approximate the optimal sampling policy; for example one might want to interrupt uninformative experiences, which do not bring any return for a while. Another place for algorithm sophistication is sample pruning for the case when the set of experiences gets large. A few most representative samples can reduce the computational cost of estimation.

Similar sampling-related problems arise in various fields; e.g., in the field of computer graphics. For a discussion of various sampling methods see dissertation of Eric Veach [181]. For a good guideline of adaptive Monte Carlo methods see section 7.8 of "Numerical Recipes in C" [132].

Chapter 6

Sample Complexity of Policy Evaluation

Summary In this chapter, we consider the question of accumulating sufficient experience for uniform convergence of policy evaluation as related to various parameters of environment and controller. We derive sample complexity bounds analogous to these used in statistical learning theory for the case of supervised learning.

6.1 Introduction

Policy search methods rely on estimating the value of policies (or the gradient of the value) at various points in a policy class and attempt to solve the *optimization issue*. Optimization algorithms work with an estimate $\hat{V}()$ of the return surface, rather than with return surface $V()$ itself. Therefore it is important to have a good estimation, formally $|\hat{V} - V| < \epsilon$, for some small ϵ and any policy. This point is illustrated in figure 6.1 (compare to figure 1.3).

In this chapter we ignore the optimization issue and concentrate on the *estimation issue*—how much and what kind of experience one needs to generate in order to be able to construct uniformly good value estimators over the whole policy class. In particular we would like to know what the relation is between the number of sample experiences and the confidence of value estimates across the policy class. We provide bounds for the uniform convergence of estimates on the policy

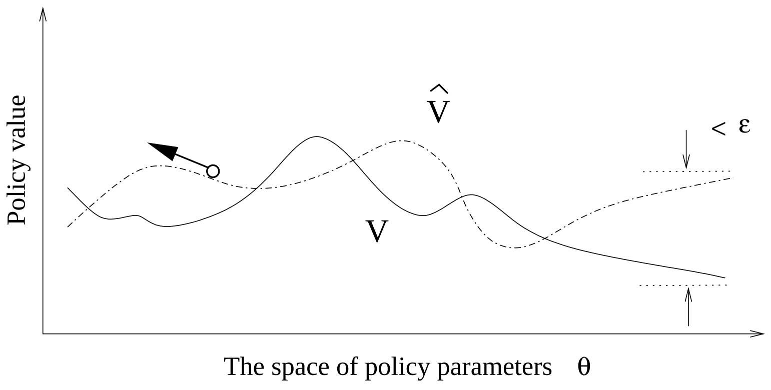


Fig. 6.1: Revisiting a problem of learning as optimization.

class. We suggest a way of using these bounds to select among candidate classes of policies with various complexities, similar to structural risk minimization [179].

To characterize the estimator we use so-called *probably approximately correct* (PAC) setting. Intuitively it means that certain assertion is likely to be close to a correct one. The PAC paradigm was introduced by Valiant [178] and developed for significant statistical learning cases by Haussler [61, 62]. For some very small values δ and ϵ, we wish to guarantee that with probability more than $(1-\delta)$ the error in the estimate of the value function is less than ϵ, formally

$$\Pr\left(\sup_{\theta\in\Theta}\left|V(\theta)-\hat{V}(\theta)\right|\leq\epsilon\right)\geq 1-\delta\ .$$

We derive bounds on the necessary sample size N, which depend on δ, ϵ, the complexity of the hypothesis class Θ expressed by the covering number $\mathcal{N}(\Theta)$, and an upper bound on the policy's absolute value $V_{\max}$. We start by reciting several important facts.

Fact 1 (Bernstein [20]) *Let* $\xi_1, \xi_2, \ldots$ *be independent random variables with identical mean* $\mathrm{E}\xi$*, bounded by some constant* $|\xi_i| \leq a$*,* $a > 0$ *. Also let* $\mathrm{Var}(M_N) \leq c$*. Then the partial sums* $M_N = \xi_1 + \ldots + \xi_N$ *obey the following inequality for all* $\epsilon > 0$*:*

$$\Pr\left(\left|\frac{1}{N}M_N - \mathrm{E}\xi\right| > \epsilon\right) \leq 2\exp\left(-\frac{1}{2}\frac{\epsilon^2 N}{c+a\epsilon}\right)\ .$$

Fact 2 (McDiarmid [99]) *Let* $X_1, \dots, X_N$ *be independent random variables taking values in a set* Ω_X*, and assume that* $f : \Omega_X^n \to R$ *satisfies*

$$\sup_{x_1,\dots,x_n,x_i' \in \Omega_X} \left| f(x_1, \dots, x_n) - f(x_1, \dots, x_{i-1}, x_i', x_{i+1}, \dots, x_n) \right| \leq c_i \ , \tag{6.1}$$

for $1 \leq i \leq n$*. Then for all* $\epsilon > 0$

$$\Pr\left(\left| f(X_1, \dots, X_n) - \mathrm{E} f(X_1, \dots, X_n) \right| \geq \epsilon \right) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n c_i^2} \right) .$$

Fact 3 (Devroye) *Let* $X_1, \dots, X_N$ *be independent random variables taking values in a set* Ω_X*, and assume that* $f : \Omega_X^n \to R$ *satisfies the condition of equation* (6.1)*. Then for all* $\epsilon > 0$

$$\mathrm{Var}\{f(X_1, \dots, X_n)\} \leq \frac{1}{4} \sum_{i=1}^n c_i^2 \ .$$

Definition 3 *Let* Θ *be a class of policies that form a metric space and let* $\varepsilon > 0$*. The covering number is defined as the minimal integer number* $\mathcal{N}(\Theta, \varepsilon)$ *of disks of radius* ε *covering* Θ *(see figure 6.2). If no such cover exists for some* $\varepsilon > 0$ *then the covering number is infinite. The metric entropy is defined as* $\mathcal{K}(\Theta, \varepsilon) = \log \mathcal{N}(\Theta, \varepsilon)$[1]*. When it does not bring ambiguity we will simply denote metric entropy* $\mathcal{K}$*.*

The rest of this chapter is organized as follows. Sections 6.2 and 6.3 present the derivation of sample complexity bounds for IS and WIS estimators correspondingly. Section 6.4 compares these bounds to similar results for related learning algorithms. Section 6.5 presents an improvement of sample complexity bounds obtained by bounding the likelihood ratio from policy class complexity considerations. Section 6.6 states open problems and provides discussion of results.

6.2 Bounds for IS Estimator

We begin by deriving bounds on the deviation of the IS estimator for a single policy from its expectation using Bernstein's inequality.

[1]According to Dudley [46], the concept of metric entropy was introduced by Kolmogorov [77, 78] as ϵ-*entropy*.

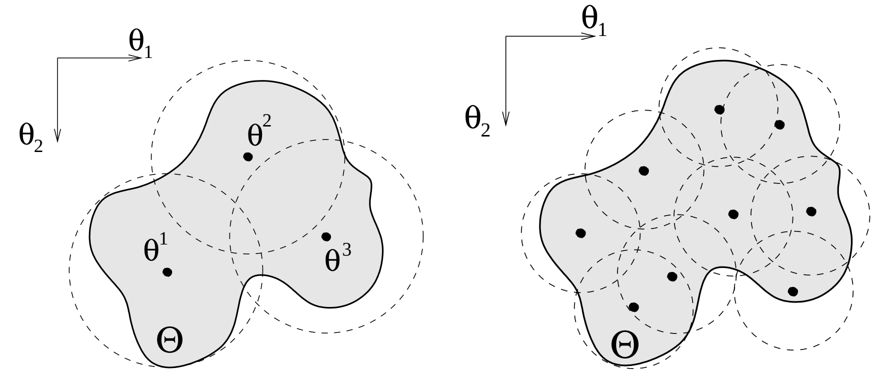


Fig. 6.2: Covering of policy space with two different cover sizes.

Lemma 6.1 *For any particular policy $\theta \in \Theta$ and the upper bound on likelihood ratio η, the estimated value $\hat{V}^{IS}(\theta)$ based on N samples is closer than ϵ to the true value $V(\theta)$ with probability at least $(1-\delta)$ if:*

$$N \quad > \quad 2\log(2/\delta)\Big(\frac{V_{\max}}{\epsilon}\Big)^2(\eta-1)\ ,$$

Proof: In our setup, $\xi_i = R(h_i)w_\theta(h_i,\theta')$, and $E\xi = E_{\theta'}\big[R(h_i)w_\theta(h_i,\theta')\big] = E_\theta\big[R(h_i)\big] = V(\theta)$; and $a = V_{\max}\eta$. According to equation (5.3) we have $c = \mathrm{Var}(M_N) = \mathrm{Var}\hat{V}^{IS}_{\theta'}(\theta) \le N\frac{V^2_{max}}{N}(\eta-1) = V^2_{max}(\eta-1)$. So we can use Bernstein's inequality to get, for a policy θ, the following deviation bound:

$$\Pr\left(\left|V(\theta)-\hat{V}^{IS}(\theta)\right| > \epsilon\right) \le 2\exp\left[-\frac{1}{2}\frac{\epsilon^2 N}{V^2_{max}(\eta-1)+V_{\max}\eta\epsilon}\right] = \delta\ . \tag{6.2}$$

After solving for N, we get the statement of Lemma 6.1.

In more detail:

$$\begin{aligned} \log(2/\delta) &= \left[\frac{1}{2}\frac{\epsilon^2 N}{V^2_{max}(\eta-1)+V_{\max}\eta\epsilon}\right] \\ N &= 2\log(2/\delta)\Big(\frac{V_{\max}}{\epsilon}\Big)\eta + 2\log(2/\delta)\Big(\frac{V_{\max}}{\epsilon}\Big)^2(\eta-1)\ , \end{aligned}$$

□

Note that this result is for a single policy. We need a convergence result *simultaneously* for all policies in the class Θ. We proceed using classical uniform convergence results for covering numbers as a measure of complexity.

Theorem 6.1 *Given a class of policies Θ with finite covering number $\mathcal{N}(\Theta, \epsilon)$, the upper bound η on the likelihood ratio $w_\theta(h_i, \theta')$, and $\epsilon > 0$, with probability at least $1-\delta$, the difference $|V(\theta)-\hat{V}^{IS}(\theta)|$ is less than ϵ simultaneously for all $\theta \in \Theta$ for the sample size*

$$N \geq 128\eta\Big(\frac{V_{\max}}{\epsilon}\Big)^2\Big(\log(8\mathcal{N}/\delta)\Big)\ .$$

Proof: In this theorem we extend equation (6.2) of lemma 6.1 from a single policy θ to the class Θ. This requires the sample size N to increase accordingly to achieve the given confidence level δ. It is accounted for by the covering number $\mathcal{N}$.

The derivation uses so-called symmetrisation and is similar to the uniform convergence result of Pollard [128](see pages 24-27), using Bernstein's inequality instead of Hoeffding's [63]. See also Chapter 4 of Bartlett's book [5] (in particular, section 4.3 "Proof of Uniform Convergence Result", pp 45-50).

$$\Pr\left(\sup_{\theta\in\Theta}\left|V(\theta) - \hat{V}^{IS}(\theta)\right| > \epsilon\right) \leq 8\mathcal{N}\left(\Theta, \frac{\epsilon}{8}\right)\exp\left[-\frac{1}{128}\frac{\epsilon^2 N}{V_{max}^2(\eta-1) + \frac{V_{\max}\eta\epsilon}{8}}\right]\ .$$

Solving for N yields the statement of the theorem.

In more detail, to solve for N, renaming $\rho = \log(8\mathcal{N}/\delta)$, we get

$$\begin{aligned}\rho &= \left[\frac{1}{128}\frac{\epsilon^2 N}{V_{max}^2(\eta-1) + \frac{1}{8}V_{\max}\eta\epsilon}\right]\\ N &= 16\Big(\frac{V_{\max}}{\epsilon}\Big)\rho\eta + 128\Big(\frac{V_{\max}}{\epsilon}\Big)^2\rho(\eta-1)\ .\end{aligned}$$

This gives us the statement of the theorem. □

Now we prove a slightly looser bound for the IS estimator using McDiarmid's theorem, with the purpose of later comparison to other bounds.

The derivation is based on the fact that replacing one history h_i in the set of samples $h_i, i \in (1..N)$ for the estimator $\hat{V}^{IS}_{\theta'}(\theta)$ of equation 5.2, cannot change the value of the estimator by more than $\frac{V_{max}\eta}{N}$.

$$f(h_1, \dots, h_N) = \hat{V}^{IS}_{\theta'}(\theta) = \frac{1}{N}\sum_i R(h_i) w_\theta(h_i, \theta').$$

$$\sup_{h_1,\dots,h_N,h_i' \in H} |f(h_1, \dots, h_N) - f(h_1, \dots, h_{i-1}, h_i', h_{i+1} \dots, h_N)| \le \frac{1}{N} V_{max}\eta \ . \tag{6.3}$$

According to McDiarmid's theorem 2, we get

$$\Pr\left(\left|V(\theta) - \hat{V}(\theta)\right| > \epsilon\right) \le 2\exp\left(-\frac{2\epsilon^2 N}{V^2_{max}\eta^2}\right) , \tag{6.4}$$

which gives us another bound,

$$N = O\left(\left(\frac{V_{max}}{\epsilon}\right)^2 \eta^2 \left(\mathcal{K} + \rho\right)\right) . \tag{6.5}$$

6.3 Bounds for WIS Estimator

In order to simplify the presentation, we begin by proving a lemma, which yields the main result. This lemma states the upper bound for the change in the value of the estimator $\hat{V}^{WIS}_{\theta'}(\theta)$ after replacing one experience h_j by another experience h_j' in the set of sample experiences $h_i, i \in (1..N)$.

Lemma 6.2 *For functions* $w_\theta(h_i, \theta')$ *and* $R(h_i)$*, such that* $0 < a \le w_\theta(h_i, \theta') \le b$ *and* $0 \le R(h_i) \le V_{max}$ *for all* $i \in (1..N+1)$ *and*

$$f(h_1, \dots, h_N) = \hat{V}^{WIS}_{\theta'}(\theta) = \frac{\sum_i R(h_i) w_\theta(h_i, \theta')}{\sum_i w_\theta(h_i, \theta')} ,$$

we have:

$$\sup_{h_1,\dots,h_N,h_j,h_j' \in H} \left|f(h_1, \dots, h_N) - f(h_1, \dots, h_{j-1}, h_j', h_{j+1} \dots, h_N)\right| \le \frac{V_{max} b}{Na + b} . \tag{6.6}$$

Proof: Let's put an upper bound on the difference:

$$\Big|f(h_1,\dots,h_N) - f(h_1,\dots,h_{j-1},h'_j,h_{j+1}\dots,h_N)\Big| =$$

$$= \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i) + w_\theta(h_j,\theta')R(h_j)}{\sum_{i\neq j} w_\theta(h_i,\theta') + w_\theta(h_j,\theta')} - \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i) + w_\theta(h'_j,\theta')R(h'_j)}{\sum_{i\neq j} w_\theta(h_i,\theta') + w_\theta(h'_j,\theta')}$$

$$\left[\text{setting } R(h_j) \text{ to } V_{\max} \text{ and } R(h'_j) \text{ to zero since they only appear once in the equation:}\right]$$

$$\leq \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i) + w_\theta(h_j,\theta')V_{\max}}{\sum_{i\neq j} w_\theta(h_i,\theta') + w_\theta(h_j,\theta')} - \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i)}{\sum_{i\neq j} w_\theta(h_i,\theta') + w_\theta(h'_j,\theta')} ,$$

$$\left[\text{the next step is clearly to set } w_\theta(h'_j,\theta') = w_\theta(h_j,\theta') = b \text{ in denominator:}\right]$$

$$\leq \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i) + bV_{\max}}{\sum_{i\neq j} w_\theta(h_i,\theta') + b} - \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i)}{\sum_{i\neq j} w_\theta(h_i,\theta') + b}$$

$$= \frac{\sum_{i\neq j} w_\theta(h_i,\theta')R(h_i) + bV_{\max} - \sum_{i\neq j} w_\theta(h_i,\theta')R(h_i)}{\sum_{i\neq j} w_\theta(h_i,\theta') + b}$$

$$= \frac{bV_{\max}}{\sum_{i\neq j} w_\theta(h_i,\theta') + b}$$

$$\left[\text{which is clearly maximized at } w_\theta(h_i,\theta') = a\right]$$

$$\leq \frac{V_{\max}b}{Na + b} .$$

□

Lemma 6.3 *Given the upper bound* η *on likelihood ratio* $w_\theta(h_i,\theta')$ *and the maximal value* $V_{\max}$,

$$\Pr\left(\sup_{\theta\in\Theta}\left|\hat{V}^{\mathrm{WIS}}(\theta) - \mathrm{E}\left[\hat{V}^{\mathrm{WIS}}(\theta)\right]\right| > \epsilon\right) \leq 8\mathcal{N}\left(\Theta, \frac{\epsilon}{8}\right)\exp\left[-\frac{\epsilon^2(N+\eta^2)^2}{32V_{max}^2\eta^4 N}\right] .$$

Proof: We may recall that in lemma 6.2, $w_\theta(h_i,\theta')$ are likelihood ratios and $R(h_i)$ are the returns of corresponding experiences h_i. For the special case of $w_\theta(h_i,\theta')$ range $b = \eta$, $a = \frac{1}{\eta}$ we get a bound: $\frac{V_{\max}\eta^2}{N+\eta^2}$. Two inequalities follow from this fact. The variance of WIS estimator according to Devroye theorem is

$$\mathrm{Var}\left\{\hat{V}^{\mathrm{WIS}}(\theta)\right\} \leq \frac{V_{max}^2\eta^4 N}{4(N+\eta^2)^2} \quad \text{or} \quad \mathrm{Var}\left\{\hat{V}^{\mathrm{WIS}}(\theta)\right\} \leq O\left(\frac{V_{max}^2}{N}\right) .$$

McDiarmid's theorem gives, for any particular policy $\theta \in \Theta$:

$$\Pr\left(\left|\hat{V}^{\mathrm{WIS}}(\theta) - \mathrm{E}\left[\hat{V}^{\mathrm{WIS}}(\theta)\right]\right| > \epsilon\right) \leq 2\exp\left[-\frac{2\epsilon^2(N+\eta^2)^2}{V_{max}^2\eta^4 N}\right] .$$

Extending this from one policy to the class Θ, analogous to the theorem 6.1 results in the statement of the lemma. □

Theorem 6.2 *Given a class of policies Θ with finite covering number $\mathcal{N}(\Theta,\epsilon)$, the upper bound η on the likelihood ratio $w_\theta(h_i,\theta')$, $\epsilon > 0$ and $\delta > 0$,*

$$N \geq 32\left(\frac{V_{\max}}{\epsilon}\right)^2 \eta^4\left(\mathcal{K}(\Theta) + \log(8/\delta)\right) .$$

Proof: Lemma 6.3 gives us an equation

$$8\mathcal{N}(\Theta,\frac{\epsilon}{8})\exp\left[-\frac{\epsilon^2(N+\eta^2)^2}{32V_{max}^2\eta^4 N}\right] = \delta .$$

Let us denote $\rho = \log\left(\frac{8\mathcal{N}}{\delta}\right)$ and solve for N to get the sample complexity bound,

$$\begin{aligned}\frac{\epsilon^2(N+\eta^2)^2}{32V_{max}^2\eta^4 N} &= \rho \\ \epsilon^2N^2 + 2\epsilon^2\eta^2N - 32V_{max}^2\eta^4\rho N + \epsilon^2\eta^4 &= 0 .\end{aligned}$$

Solving the quadratic equation with respect to N,

$$\begin{aligned}N &= \frac{1}{2\epsilon^2}\left(-2\epsilon^2\eta^2 + 32V_{max}^2\eta^4\rho + \sqrt{4\epsilon^4\eta^4 + 2^{10}V_{max}^4\eta^8\rho^2 - 2^7\epsilon^2\eta^6\rho - 4\epsilon^4\eta^4}\right) \\ &= \frac{1}{\epsilon^2}\left(-\epsilon^2\eta^2 + 16V_{max}^2\eta^4\rho + 4\eta^3\sqrt{\left(16V_{max}^4\eta^2\rho - \epsilon^2\right)\rho}\right) \\ &\geq \frac{1}{\epsilon^2}\left(-\epsilon^2\eta^2 + 32V_{max}^2\eta^4\rho\right) \\ &\geq 32\left(\frac{V_{\max}}{\epsilon}\right)^2\eta^4\rho .\end{aligned}$$

□

6.4 Comparison to VC Bound

In the pioneering work by Kearns *et al.* [73], the issue of generating enough information to determine the near-best policy is considered.

Tab. 6.1: Comparison of sample complexity bounds.

	Sample complexity N
IS	$128\left(\frac{V_{\max}}{\epsilon}\right)^2 2^T(1-\underline{c})^T\left(\mathcal{K}(\Theta)+\log(8/\delta)\right)$
WIS	$32\left(\frac{V_{\max}}{\epsilon}\right)^2 2^{4T}(1-\underline{c})^{4T}\left(\mathcal{K}(\Theta)+\log(8/\delta)\right)$
VC	$\left(\frac{V_{\max}}{\epsilon}\right)^2 2^{2T}VC(\Theta)\left(T+\log\left(\frac{V_{\max}}{\epsilon}\right)+\log(1/\delta)\right)\log(T)$

Let us compare our sample complexity results with a similar result for their *"reusable trajectories"* algorithm. Using a random policy (selecting actions uniformly at random), this algorithm generates a set of history trees. This information is used to define estimates that uniformly converge to the true values. However, this work relies on having a generative model of the environment, which allows simulation of a reset of the environment to any state and the execution of any action to sample an immediate reward. Also, the reuse of information is partial—an estimate of a policy value is built only on a subset of experiences, "consistent" with the estimated policy.

We will make a comparison based on a sampling policy that selects one of two actions uniformly at random: $\Pr\left(a|h\right)=\frac{1}{2}$. For the horizon T, this gives us an upper bound η on the likelihood ratio:

$$w_\theta(h,\theta') \leq 2^T(1-\underline{c})^T = \eta\ . \tag{6.7}$$

Substituting expression for η, we can compare our bounds to the bound of Kearns *et al.* as presented in table 6.1. All bounds are exponential in the horizon, however in the last case, the dependence on $\left(\frac{V_{\max}}{\epsilon}\right)$ has an extra logarithmic term rather than quadratic. Interestingly, the same term comes out if we notice that covering a range of 0 to V_{max} by intervals of size ϵ gives a covering number $\mathcal{N}=\left(\frac{\mathcal{V}_{\max}}{\epsilon}\right)$ and a corresponding metric entropy $\mathcal{K}=\log\left(\frac{V_{\max}}{\epsilon}\right)$. It is logical to expect some reduction in sample size for IS algorithm compared to *"reusable trajectories"*, since the former uses all trajectories for evaluation of any policy, while the latter uses just a subset of the trajectories. The factor of reduction then has to do with the fraction of samples from the unique sampling policy which could be reused for evaluating other policies—a combinatorial quantity directly related to a dimension of policy space

$\mathrm{VC}(\Theta)$. Contrary to our intuition, the bound for IS is tighter than the one for WIS estimator. This could be attributed to using McDiarmid's equation in the derivation. The metric entropy $\mathcal{K}(\Theta)$ takes the place of the VC-dimension $\mathrm{VC}(\Theta)$ (like in Kearns *et al.* [73]) in terms of policy class complexity. Metric entropy is a more refined measure of capacity than VC-dimension since the VC-dimension is an upper bound on the growth function which is an upper bound on the metric entropy [179]. We use metric entropy both in a union bound and as a parameter for bounding the likelihood ratio in the next section.

6.5 Bounding the Likelihood Ratio

In this section we show that if we are working with a policy class of a limited complexity, the likelihood ratio can be bounded above through the covering number, due to the limit in combinatorial choices. Remember that we are free to choose a sampling policy. The goal is to find a sampling policy that minimizes sample complexity of policy evaluation on a given policy class. We have discussed what it means for one sampling policy to be optimal with respect to another (see page 77). Here we would like to consider what it means for a sampling policy θ to be optimal with respect to a policy class Θ. The intuition is that we want to minimize the maximal possible difference in likelihoods of any experience under our sampling policy and any other policy from a given policy class. This means that the sampling policy is optimal in the information-theoretic sense. Choosing the optimal sampling policy allows us to improve bounds with regard to exponential dependence on the horizon T.

The derivation is very similar to the one of an upper bound on the minimax regret for predicting probabilities under logarithmic loss [34, 114]. The upper bounds on logarithmic loss we use were first obtained by Opper and Haussler [114] and then generalized by Cesa-Bianchi and Lugosi [34]. The result of Cesa-Bianchi and Lugosi is more directly related to the reinforcement learning problem since it applies to the case of arbitrary rather than *static* experts, which corresponds to learning a policy. First, we describe the sequence prediction problem and result of Cesa-Bianchi and Lugosi, then show how to use this result in our setup.

Sequential Prediction Game

With some abuse of notation, we deliberately describe a sequential prediction game in the same notation as reinforcement learning problem to make explicit semantic parallels. In a sequential prediction game, T symbols $h_a^T = \langle a(1), \dots, a(T) \rangle$ are observed sequentially. After each observation $a(t-1)$, a learner is asked how likely it is for each value $a \in A$ to be the *next* observation. The learner's goal is to assign a probability distribution $\Pr(a(t)|h_a^{t-1};\theta')$ based on the previous values. When at the next time step t, the actual new observation $a(t)$ is revealed, the learner suffers a loss of $-\log(\Pr(a(t)|h_a^{t-1};\theta')$. At the end of the game, the learner has suffered a total loss of $-\sum_{t=1}^{T} \log \Pr(a(t)|h_a^{t-1};\theta')$. Using the joint distribution $\Pr(h_a^T|\theta') = \prod_{t=1}^{T} \Pr(a(t)|h_a^{t-1};\theta')$ we are going to write the loss as $-\log \Pr\left(h_a^T|\theta'\right)$.

When it is known that the sequences h_a^T are generated by some probability distribution θ from the class Θ, we might ask what is the worst *regret*: the difference in the loss between the learner and the best expert in the target class Θ on the worst sequence:

$$L_T = \inf_{\theta'} \sup_{h_a^T} \left\{ -\log \Pr(h_a^T|\theta') + \sup_{\theta\in\Theta} \log \Pr(h_a^T|\theta) \right\}.$$

Using the explicit solution to the minimax problem due to Shtarkov [153], Cesa-Bianchi and Lugosi prove the following theorem.

Fact 4 (Cesa-Bianchi and Lugosi [34] (theorem 3)) *For any policy class Θ:*

$$L_T \leq \inf_{\epsilon>0} \left(\log \mathcal{N}(\Theta, \epsilon) + 24 \int_0^{\epsilon} \sqrt{\log \mathcal{N}(\theta, \tau)} d\tau \right) .$$

It is now easy to relate the problem of bounding the likelihood ratio to the worst case regret. Intuitively, we are asking what is the worst case likelihood ratio if we have the optimal sampling policy. Optimality means that our sampling policy will induce action sequences with probabilities close to the policies, whose values we want to estimate. Remember that the likelihood ratio depends only on the action sequence h_a in the history h (see Lemma 5.1). We need to provide an upper bound on the maximum value of the ratio $\frac{\Pr(h_a|\theta)}{\Pr(h_a|\theta')}$, which corresponds to $\inf_{\theta'} \sup_{h_a} \left(\frac{\Pr(h_a|\theta)}{\Pr(h_a|\theta')}\right)$.

Lemma 6.4

$$\inf_{\theta'} \sup_{h_a} \left(\frac{\Pr(h_a|\theta)}{\Pr(h_a|\theta')} \right) \leq \inf_{\theta'} \sup_{h_a} \left(\frac{\sup_{\theta \in \Theta} \Pr(h_a|\theta)}{\Pr(h_a|\theta')} \right) .$$

Proof: The statement of lemma immediately follows by the definition of the maximum likelihood policy $\sup_{\theta \in \Theta} \Pr(h_a|\theta)$. □

Henceforth we can directly apply the results of Cesa-Bianchi and Lugosi and get a bound of $\eta = \exp(L_T)$. Note the logarithmic dependence of the bound on L_T with respect to the covering number $\mathcal{N}$. Moreover, since actions a belong to the finite set of actions A, many of the remarks of Cesa-Bianchi and Lugosi regarding finite alphabets apply [34]. In particular, for most "parametric" classes—which can be parametrized by a bounded subset of $\mathcal{R}^n$ in some "smooth" way [34]—the metric entropy scales as follows: for some positive constants k_1 and k_2,

$$\log \mathcal{N}(\Theta, \epsilon) \leq k_1 \log \frac{k_2 \sqrt{T}}{\epsilon} .$$

For such policies the minimax regret can be bounded by

$$L_T \leq \frac{k_1}{2} \log T + o(\log T) ,$$

which makes the likelihood ratio bound of

$$\eta \leq O\left(T^{\frac{k_1}{2}}\right) .$$

The sample complexity bound for the IS estimator becomes

$$N \geq 128 \left(\frac{V_{\max}}{\epsilon} \right)^2 T^{\frac{k_1}{2}} \left(\log \left(\frac{V_{\max}}{\epsilon} \right) + \log(8/\delta) \right) .$$

To conclude, there are such policy classes for which an exponential dependence of the sample complexity on the horizon is eliminated, assuming we know how to find the corresponding optimal sampling policy. However, it remains an open problem to estimate a constant k_1 and to obtain a constructive solution to the estimation problem. Namely, to find a way of choosing a sampling policy which minimizes the sample complexity.

6.6 Discussion and Open Problems

In this chapter, we considered the question of accumulating sufficient experience and gave sample complexity bounds for policy evaluation using likelihood ratio estimation. Note that for these bounds to be meaningful, the covering number $\mathcal{N}(\Theta)$ of the class of policies Θ should be finite. We also believe that with more rigorous analysis some constants in our bounds could be improved.

We have already discussed the relation of this work to work by Kearns *et al.* in section 6.4. Mansour [91] has addressed the issue of computational complexity in the setting of Kearns *et al.* [73] by establishing a connection between *mistake bounded algorithms* (adversarial on-line model [85]) and computing a near-best policy from a given class with respect to a finite-horizon return. Access to an algorithm that learns the policy class with some maximal permissible number of mistakes is assumed. This algorithm is used to generate "informative" histories in the POMDP, following various policies in the class, and determine a near-optimal policy. In this setting a few improvements in bounds are made. Glynn *et al.* [54] provide the Hoeffding like deviation bound for ergodic Markov chain in undiscounted reward setup. Van-Roy in his dissertation provides sample complexity bounds for uniform convergence over the state space [137] for learning with value search methods.

Sample complexity bounds (theorems 6.1 and 6.2) derived in this chapter depend on the covering number. It remains an open problem to estimate and bound covering number for various controllers, in particular for FSCs. As has been suggested by Peter Bartlett the derivation is likely to follow similar work for recurrent neural networks [43]. Once the covering number has been established, we are ready to answer a very important question of how to choose among several candidate policy classes.

Our reasoning here is similar to that of the structural risk minimization principal by Vapnik [179]. The intuition is that given very limited data, one might prefer to work with a primitive class of hypotheses with high confidence, rather than getting lost in a sophisticated class of hypotheses due to low confidence. Formally, we would have the following method: given a set of policy classes $\Theta_1, \Theta_2, \ldots$ with corresponding covering numbers $\mathcal{N}_1, \mathcal{N}_2, \ldots$, a confidence δ and a number of available samples N, compare error bounds $\epsilon_1, \epsilon_2, \ldots$ according to theorem 6.1 or 6.2.

Another way to use sample complexity results is to find what is the minimal experience necessary to be able to provide the estimate for any policy in the class with a given confidence. This also provides insight for a new optimization technique. Given the value estimate, the number of samples used, and the covering number of the policy class, one can search for optimal policies in a class using a new cost function $\hat{V}(\theta) + \Phi(\mathcal{N}, \delta, N) \leq V(\theta)$. This is similar in spirit to using structural risk minimization instead of empirical risk minimization.

The capacity of the class of policies is measured by bounds on covering numbers in our work or by VC-dimension in the work of Kearns *et al.* [73]. The worst case assumptions of these bounds often make them far too loose for practical use. An alternative would be to use more empirical or data dependent measures of capacity; *e.g.*, the empirical VC-dimension [180] or maximal discrepancy penalties on splits of data [12], which tend to give more accurate results.

The fundamental idea behind reinforcement learning is that the agent is only concerned with environment's dynamics as far as determined by reinforcement. Once we have chosen the policy space to search in learning, we want to cover this space by trying various policies. But it is impossible to try out all policies in a continuous space. Instead several policies would represent well all possibilities of control.

Rather than simply estimate the covering number for a particular policy class, it would be desirable to find some kind of constructive solution in a sense of universal prediction theory [100, 153]. That would mean to solve a problem of which policies to try to cover the space of policies with a minimal number of ϵ-disks (see figure 6.2). One can imagine that for a given covering number there are several ways to construct the actual covering set.

For the distance $D(\theta, \theta') \doteq |V(\theta) - V(\theta')|$ and centers of the covering balls $\theta^1 ... \theta^{\mathcal{N}}$, make the cover set $\mathcal{C}$. Let us define the function $\mathrm{cov}(\theta, \mathcal{C})$ as a center of the cover ball closest to θ. For our application we might require the cover that minimizes

$$\min_{\mathcal{C}} \int_{\Theta} D\Big(\theta, \mathrm{cov}(\theta, \mathcal{C})\Big) .$$

There might be other meaningful ways to quantify the complexity of an RL problem. We want to characterize the complexity of the policy space with regard to returns on experiences, policies from this space could induce in the environment/agent interaction process. The complexity of the environment dynamics could be defined as entropy (see for example the paper by Crutchfield [42]).

In his manuscript "Open theoretical questions in reinforcement learning" Sutton suggests that "Recently, some progress has been made by Kearns, Mansour and Ng [73] that seems to open up a whole range of new possibilities for applying COLT ideas to RL.". He further notes that "the conventional definition of VC-dimension cannot be directly applied to policy sets". Sutton proposed that finding "some kind of VC-dimension for reinforcement learning" is an interesting open problem [166]. We believe that this work constitutes a step in this direction and provides a clearer formulation of this problem.

Eſ ſteigt, eſ blitzt, eſ hauft ſich an,
Im Augenblick iſt eſ getan.
Ein großer Vorſatz ſcheint im Anfang toll;
Doch wollen wir deſ Zufallſ kunftig lachen,
Und ſo ein Hirn, daſ trefflich denken ſoll,
Wird kunftig auch ein Denker machen.[1]

Chapter 7

Conclusions

Future Research: Symbiotic intelligence

This dissertation is in the field of reinforcement learning, which is a part of statistical artificial intelligence. With apprehension or excitement, people often ask when artificial intelligence will overpower natural intelligence. Forced to make a futuristic statement, I suggest there will be no competition. Rather, I foresee the emergence of a "symbiotic intelligence". This vision, while naive, has a serious counterpart which underlies several projects on my future research agenda.

In nature, learning often happens by imitation rather than understanding. An ability to track the sensory input and to witness

[1] "It rises, flashes, gathers on; A moment, and the deed is done. A great design at first seems mad; but we Henceforth will laugh at chance in procreation, And such a brain that is to think transcendently Will be a thinker's own creation. Goethe [182].

human reactive behavior opens an avenue for "artificial intelligence by mimicking". The biology of modern man is routinely augmented by various silicon-controlled devices. Whether smart cardiostimulators, hearing aids, palm pilots or cellular phones, such devices share our physical realm and enlarge the domain of the human Ego. *How could an embodied agent learn from instances of human behaviors?* I would like to research how to form the initial behavior of an agent using supervised learning techniques and later fine-tune the behavior by RL. When using classical value-search methods (based on dynamic programming principles), an agent forgets the knowledge from supervised learning stage [167], before proceeding to improvement via reinforcement learning. This dissertation focuses on a set of novel techniques [189, 9] which directly search the space of behaviors. For these techniques, initiation of policy parameters via supervised learning is an open problem.

Humans and computers interact on equal grounds in artificial environments, such as Internet auctions, game servers and virtual places [176]. In such environments we bypass the engineering complexity of operating in physical reality with the risks of faulty motors and imperfect sensors. Learning algorithms and data structures can be verified and mastered in a digital world still directly bearing on the challenges of everyday applications and enabling electronic commerce. For example, trade-bots act or assist in purchasing goods on the Internet (see e.g. [171]). One challenge in the domain of financial applications is to extend the notion of risk-sensitive planning for Markov environments [38, 35] to partially observable environments (POMDPs). Simply optimizing the expected return is not always appropriate in such domains. Another challenge that attracts me to this application is a necessity to learn in continuous state and action spaces (such as price). It requires to answer one of the open questions in RL: how to learn in conjunction with function approximation [166, 115].

Methods of re-using the experience for learning would address the central issues in RL in general and in direct policy search methods in particular: a vast amount of experience is necessary for learning, and a high variance of estimates resulting in slow convergence. Establishing bounds for the convergence properties of various learning algorithms has been a recent focus in RL community [91, 73, 96, 16]. In this dissertation we explored the use of likelihood ratio (importance sampling) techniques [103, 124] adapted from stochastic optimization theory. Another direction is in partitioning the state space into several domains according to behavior [57, 56, 64] and then adjusting the borderlines. This state grouping could be viewed in light of learning a feature extraction mechanism.

Learning becomes easy once the right set of features is in place. All problems computers are expected to solve, whether in digital or physical realms, are formulated by humans and are solvable by humans. This implies that there is a language or representation, which allows for efficient solutions. In nature, this key property is a result of millennia-long selection by evolutionary process; which enables living systems to learn within a short lifespan. *It is necessary for human designers to parallel the job of evolution—engineer and supply a set of features.* Ii is important to examine a process of feature design and discovery from biological and computational points of view [140, 84, 144]. In particular, I would be interested in building on my experience in computer vision and machine learning in order to work on learning for visual control systems [26, 145] and active vision systems [1, 10, 7, 3]. These systems are used for visual search and scene analysis. A control problem is to choose which of several visual routines [177] to invoke under the constraint of limited computational and optical resources. A similar exciting problem is modeling human linguistic behavior in an effort to de-formalize the protocol of human-computer interaction [72]. It is important to research ways of incorporating the knowledge of the syntactic structure of natural language into an AI agent, enabling it to learn semantic features of text.

Statistical learning theory provides a unified approach to image and text processing [179] We have presented a step in the direction of extension of computational learning theory to reinforcement learning and hypothesis testing in RL. The ultimate goal is to devise a principle, similar to *structural risk minimization* [179], of choosing among several possible architectures of controllers [124] according to the size of the available experience. Particularly intriguing would be to derive a covering number for the class of finite state controllers [123, 104].

Learning in multi-agent systems has been a long standing goal in RL *[37, 24, 66].* It is related to a well established idea [105, 14] that intelligent behavior is a result of cooperation in a society of (distributed) primitive agents. There is an excellent opportunity to extend the part of this dissertation on learning to cooperate [122] and adaptive routing in static network topologies onto adaptive control in wireless communication and *ad-hoc* networks, e.g. to simultaneously maximize channel utility and battery life, maintain quality of service or connectivity patterns in multi-hop add-hock networks. RL has already been applied to find effective policies within the dynamics of telecommunication problems: e.g., channel allocation in wireless systems, network routing, and admission control [25, 93, 156, 28, 30].

Potentially utilizing outcomes of all aforementioned research, perhaps the most distant and ambitious project is to design principles for a brain/machine interface, and to develop ideas about how the nervous system dynamically organizes representations that may be used to control neurosprosthetic interfaces [90]. This would be truly a step on the path to symbiotic intelligence.

Contributions

The research presented in this dissertation constitutes a number of contributions to the field of constructing an adaptive system which learns from a feedback signal in partially observable environments. These contributions[2] are outlined below in the order in which they appear in the dissertation.

The description of the reinforcement learning problem brings together methods from dynamic programming that are based on the concept of state-action value with methods closely related to stochastic optimization and control that are based on direct search in policy space. The overview of existing work in the field of reinforcement learning by policy search provides a unified perspective on various directions in constraining the search space and ways to direct the search.

The development of a gradient ascent algorithm for the case of reactive policies working with the augmented observation and action spaces is new. In particular we gave a clear semantic interpretation of the portions of reinforcement assigned to each parameter responsible for a particular observation-action pair in the policy encoding.

Original contributions are presented in the adaptation of policy learning methods to the case of finite-state controllers for partially observed problems as well as testing on a number of domains including partially observable pole balancing.

[2]Subject to the remarks made in the "Publication notes" (p. vii)

In the area of multi-agent learning, contributions include: developing a gradient descent algorithm for multiple controllers with memory; relating local optima in policy space to the game-theoretic solution concept of Nash equilibrium; empirical results for a small simulated soccer domain; an application to a packet routing problem.

Stochastic optimization algorithms used in reinforcement learning rely on estimates of the value of a policy. Typically, the value of a policy is estimated from results of simulating that very policy in the environment. This approach requires a large amount of simulation as different points in the policy space are considered. The adaptation of value estimators that use data gathered from following one policy to estimate the value of another policy, for some domains resulting in much more data-efficient algorithms, is new. While likelihood-ratio estimators are widely used in optimization problems, in policy search methods for reinforcement learning they were introduced only recently.

The question of accumulating sufficient experience for uniform convergence of policy evaluation as related to various parameters of environment and controller constitutes original research. The sample complexity bounds derived here are an original contribution.

Finally, we regard as a valuable contribution the rigorous formulation of problems in the area of quantifying the complexity of a policy space as related to statistical learning theory results and the discussion of other open problems and directions for further research, which will serve as a solid starting point for forthcoming dissertations.

List of Figures

List of Tables

Reinforcement Learning tramite ricerca di strategie

Tesi di dottorato di Leon Peshkin

Uno degli obiettivi dell'intelligenza artificiale consiste nel modellare il comportamento di un agente intelligente che interagisca con il suo ambiente. Le trasformazioni dell'ambiente possono essere modellate mediante una catena Markoviana i cui stati siano parzialmente osservabili dall'agente e dipendenti dalle sue azioni; tali processi sono noti come processi di decisione Markoviana parzialmente osservabili (POMDP: partially observable Markov decision process). Si assume che la dinamica dell'ambiente sia determinata da un insieme di regole che l'agente non conosce e deve imparare. In questa tesi ci concentriamo sullo studiare come modellare, mediante il meccanismo del reinforcement learning, la capacità di adattamento dell'agente. Reinforcement learning significa imparare una strategia - associare azioni a osservazioni - sulla base delle risposte ricevute dall'ambiente. L'apprendimento può essere visto come la scelta tra un insieme di strategie la cui bontà è valutata basandosi su esperimenti fatti interagendo con l'ambiente. L'insieme delle strategie possibili è vincolato dall'architettura del controllore dell'agente. I POMDP richiedono che il controllore sia dotato di memoria. Tra le varie tipologie di controllori con memoria presi in considerazioni si ricordano i controllori con memoria esterna, i controllori a stati finiti e i controllori distribuiti per sistemi multi-agente. Per ognuna di queste categorie forniremo i dettagli degli algoritmi che permettono di imparare risalendo il gradiente del rinforzo cumulativo atteso. Sulla base della teoria dell'apprendimento statistico e della teoria di progettazione degli esperimenti viene sviluppato un algoritmo di valutazione delle strategie nel caso di riutilizzo delle esperienze. Affrontiamo inoltre il problema di determinare la quantità di esperienze necessaria a garantire la convergenza uniforme nella valutazione delle strategie ottenendo dei bound per diversi stimatori. Infine, dimostriamo l'efficienza degli algoritmi proposti in numerosi domini applicativi il più complesso dei quali è il routing adattativo di pacchetti in una rete di telecomunicazioni.

Parole chiave: MDP, POMDP, scelta di strategie, metodi con gradiente, reinforcement learning, sistemi adattativi, controllo stocastico, comportamento adattativo.

Теория Поиска Стратегии в Компутерном Обучении с Поощрением

Рефферат диссертации Леонида Пешкина, 2002
Массачусеттский Технологический Институт, США

Одной из основных задач в области искусственного интеллекта является построение формальной модели взаимодействия интеллектуального агента со средой его обитания. Процесс преобразований состояния среды может быть представлен в виде цепи Маркова. Состояние цепи при этом только частично наблюдаемо агентом. Действия агента влияют на динамику переходов среды из текушего состоыания в следуюшее. Динамика среды подчиняется определенным правилам, неизвестным обучаюшемуся агенту. Эта модель получила название "процесс Принятия Решений в Частично наблюдаемой цепи Маркова" (ПРЧМ).

Данная диссертация представляет вклад в область адаптирующихся агентов в рамках теории обучения с поощрением. Обучение с поощрением это обучение стратегии поведения — то есть функции определяющей соответствие деиствий наблюдениям — на основе внешнего поощрительного сигнала. Это промежуточная форма обучения между "обучением с учителем" и "обучением без учителя". Обучение с поощрением можно представить как поиск оптимальной стратегии среди множества доступных стратегий, производимый путем наблюдения и взаимодействия со средой. Фактически – методом проб и ошибок.

Множество стратегий агента определяеця архетектурой регулятора агента. В рассматриваемой модели (ПРЧМ) в регуляторе необходимо наличие памяти. В данной дисертации исследуюця различные подходы к созданию регуляторов с памятью. Рассмотрены как регуляторы с внешней памятью, так и регуляторы – конечные автоматы, и распределенные регуляторы мулти-агентных систем. Для рассмотреных типов регуляторов разработаны детальные алгоритмы обучения, основанные на методах оптимизации путем градиентного спуска на фунции усредненного сигнала поощрения.

Кроме того, разработаны методы оценки стратегии с повторным использованием полученого опыта, основанные на статистической теории обучения и теории планирования експериментов. Рассмотрен вопрос о размере выборки данных необходимых для однородной сходимости оценок стратегий на всем доступном классе. Для различных способов оценки получена форма зависимости требуемого размера выборки данных от параметров конкретной среды. В заключение еффективность разработанных алгоритмов продемонстрирована в приложении к ряду конкретных задач. В частности к задаче адаптивной маршрутизации пакетов в симулированой среде передачи данных.

Bibliography

[1] J.Y. Aloimonos, I. Weiss, and Dana A. Bandopadhay, *Active vision*, International Journal on Computer Vision (1987), 333–356.

[2] Shun-ichi Amari, *Natural gradient works effciently in learning*, Neural Computation **10** (1998), 251–276.

[3] Claus Siggaard Andersen, *A framework for control of a camera head*, Ph.D. thesis, Aalborg University, Uppsala, Sweden, January 1996.

[4] Charles W. Anderson, *Learning to control an inverted pendilum using neural networks*, IEEE Control Systems Magazine **9** (1989), no. 3, 31–37.

[5] Martin Anthony and Peter Bartlett, *Neural networks learning: Theoretical foundations*, Cambridge University Press, Cambridge, UK, 1999.

[6] Kenneth J. Arrow and Leonid Hurwicz, *Stability of the gradient process in n-person games*, Journal of the Society for Industrial and Applied Mathematics **8** (1960), no. 2, 280–295.

[7] Minoru Asada, Takayuki Nakamura, and Koh Hosoda, *Purposive behavior acquisition for a real robot by vision-based reinforcement learning*, Machine Learning **23** (1996), 1–40.

[8] Leemon C. Baird, *Reinforcement learning through gradient descent*, Ph.D. thesis, CMU, Pittsburgh, PA, 1999.

[9] Leemon C. Baird and Andrew W. Moore, *Gradient descent for general reinforcement learning*, Advances in Neural Information Processing Systems, vol. 11, The MIT Press, 1999.

[10] Dana Ballard, *Animate vision*, Artificial Intelligence **48** (1991), no. 1, 1–27.

[11] Peter Bartlett and Jonathan Baxter, *Hebbian synaptic modifications in spiking neurons that learn*, Tech. report, Australian National University, Sydney, Australia, 1999.

[12] Peter Bartlett, S. Boucheron, and Gábor Lugosi, *Model selection and error estimation*, Proceedings of the Thirteenth Annual Conf. on Computational Learning Theory (New York, NY), ACM Press, 2000.

[13] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems*, IEEE Transactions on Systems, Man, and Cybernetics **SMC-13** (1983), no. 5, 834–846.

[14] Eric Baum, *Neural networks and machine learning*, ch. Manifesto for an Evolutionary Economics of Intelligence, pp. 285–344, Springer-Verlag, 1998.

[15] ______, *Toward a model of intelligence as an economy of agents*, Machine Learning **35** (1999), 155–185.

[16] Jonathan Baxter and Peter Bartlett, *Reinforcement learning in POMDP's via direct gradient ascent*, Proceedings of the Seventeenth International Conf. on Machine Learning, Morgan Kaufmann, 2000.

[17] R. Beckers, Owen E. Holland, and J. L. Deneubourg, *From local actions to global tasks: Stigmergy and collective robotics*, Artificial Life IV (Cambridge, MA), The MIT Press, 1994.

[18] Richard Bellman, *Dynamic programming*, Princeton University Press, Princeton, NJ, 1957.

[19] Daniel Bernstein, Shlomo Zilberstein, and Neil Immerman, *The compplexity of decentralized control of Markov decision processes*, Proceedings of the Sixteenth Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 2000.

[20] Sergei Natanovich Bernstein, *The theory of probability*, Gostehizdat, Moscow, Russia, 1946.

[21] Dimitri P. Bertsekas, *Dynamic programming and optimal control*, Athena Scientific, Belmont, MA, 1995, Volumes 1 and 2.

[22] Dimitri P. Bertsekas and John N. Tsitsiklis, *Neuro-dynamic programming*, Athena Scientific, Belmont, MA, 1996.

[23] Lashon B. Booker, D.E. Goldberg, and J.H. Holland, *Classifier systems and genetic algorithms*, Artificial Intelligence **40** (1989), 235–282.

[24] Craig Boutilier, *Sequential optimality and coordination in multiagent systems*, Proceedings of the Sixteenth International Joint Conf. on Artificial Intelligence, 1999, pp. 478–485.

[25] Justin Boyan and Michael L. Littman, *Packet routing in dynamically changing networks: A reinforcement learning approach*, Advances in Neural Information Processing Systems, vol. 7, The MIT Press, 1994, pp. 671–678.

[26] Rodney A. Brooks, Anita M. Flynn, and Thomas Marill, *Self calibration of motion and stereo vision for mobile robot navigation*, Tech. Report AIM-984, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1987.

[27] Timothy X. Brown, *Low power wireless communication via reinforcement learning*, Advances in Neural Information Processing Systems, vol. 12, The MIT Press, 1999, pp. 893–9.

[28] Timothy X. Brown, H. Tong, and Satinder P. Singh, *Optimizing admission control while ensuring quality of service in multimedia networks via reinforcement learning*, Advances in Neural Information Processing Systems, vol. 12, The MIT Press, 1999, pp. 982–8.

[29] J. Bucklew, *Large deviation techniques in decision, simulation, and estimation*, John Wiley, New York, NY, 1990.

[30] Jakob Carlstrom, *Reinforcement learning for admission control and routing*, Ph.D. thesis, Uppsala University, Uppsala, Sweden, 2000.

[31] G. Di Caro and M. Dorigo, *Ant colonies for adaptive routing in packet-switched communications networks*, Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature, September 1998.

[32] Anthony R. Cassandra, *Exact and approximate algorithms for partially observable Markov decision processes*, Ph.D. thesis, Brown University, Providence, RI, 1998.

[33] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman, *Acting optimally in partially observable stochastic domains*, Proceedings of the Twelfth National Conf. on Artificial Intelligence (Seattle, WA), 1994.

[34] Nicolò Cesa-Bianchi and Gábor Lugosi, *Worst-case bounds for the logarithmic loss of predictors*, Machine Learning **43** (2001), no. 3, 247–264.

[35] Jay P. Chawla, *Optimal risk sensitive control of semi-Markov decision processes*, Ph.D. thesis, The University of Maryland, Baltimore, MD, 2000.

[36] I. Cidon, S. Kutten, Y. Mansour, and D. Peleg, *Greedy packet scheduling*, SIAM Journal on Computing **24** (1995), no. 1, 148–157.

[37] Caroline Claus and Craig Boutilier, *The dynamics of reinforcement learning in cooperative multiagent systems*, Proceedings of the Tenth Innovative Applications of Artificial Intelligence Conference (Madison, WI), July 1998, pp. 746–752.

[38] S.P. Coraluppi and S.I. Marcus, *Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes*, Automatica **35** (1999), 301–309.

[39] N. L. Cramer, *A representation for the adaptive generation of simple sequential programs*, Proceedings of an International Conference on Genetic Algorithms and Their Applications (Hillsdale, NJ) (J.J. Grefenstette, ed.), Lawrence Erlbaum Associates, 1985.

[40] Robert H. Crites and Andrew G. Barto, *Improving elevator performance using reinforcement learning*, Advances in Neural Information Processing Systems, vol. 9, The MIT Press, 1997.

[41] ———, *Elevator group control using multiple reinforcement learning agents*, Machine Learning **33** (1998), 235–42.

[42] James P. Crutchfield and David P. Feldman, *Synchronizing to the environment: Information theoretic constraints on agent learning*, http://www.santafe.edu/sfi/publications/working-papers.html, 2001, manuscript.

[43] Bhaskar Dasgupta and Eduardo D. Sontag, *Sample complexity for learning recurrent perceptron mappings*, Advances in Neural Information Processing Systems (David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, eds.), vol. 8, The MIT Press, 1996, pp. 204–210.

[44] Peter Dayan and L.F. Abbott, *Theoretical neuroscience*, The MIT Press, Cambridge, MA, 2001.

[45] E. Dijkstra, *A note on two problems in connection with graphs*, Numerical Mathematics **1** (1959), 269–271.

[46] Richard M. Dudley, *Uniform central limit theorems*, Cambridge University Press, Cambridge, MA, 1999.

[47] S. Ermakov, *The Monte Carlo method and related problems*, Nauka, Moscow, 1971, (in Russian).

[48] S. Ermakov and G. Mikailov, *Statisticheskoje modelirovanie*, Nauka, Moscow, 1982, (in Russian).

[49] E. Gelenbe, Ricardo Lent, and Zhiguang Xu, *Design and analysis of cognitive packet networks*, Performance Evaluation (2001), 155–76.

[50] Peter W. Glynn, *Optimization of stochastic systems*, Proceedings of the Winter Simulation Conference, 1986.

[51] ———, *Importance sampling for stochastic simulations*, Management Science **35** (1989), no. 11, 1367–92.

[52] ———, *Likelihood ratio gradient estimation for stochastic systems*, Communications of the ACM **33** (1990), no. 10, 75–84.

[53] ———, *Importance sampling for Markov chains: Asymptotics for the variance*, Commun. Statist. – Stochastic Models **10** (1994), no. 4, 701–717.

[54] Peter W. Glynn and Dirk Ormoneit, *Hoeffding's inequality for uniformly ergodic Markov chains*, Statistical Science **35** (2001), no. 11, 1367–92.

[55] Faustino Gomez and Risto Miikkulainen, *2-d pole balancing with recurrent evolutionary networks*, Proceedings of the Intl Conf on Artificial Neural Networks, Elsevier, 1998, pp. 758–763.

[56] G. Z. Grudic and L. H. Ungar, *Localizing policy gradient estimates to action transitions*, Proceedings of the Seventeenth International Conf. on Machine Learning, Morgan Kaufmann, 2000.

[57] ———, *Localizing search in reinforcement learning*, Proceedings of the Seventeenth National Conf. on Artificial Intelligence, 2000.

[58] E.A. Hansen, *Finite-memory control of partially observable systems*, Ph.D. thesis, University of Massachusetts at Amherst, Amherst, MA, 1998.

[59] ———, *Solving POMDPs by searching in policy space*, Proceedings of the Fourteenth Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1998, pp. 211–219.

[60] Milos Hauskrecht, *Planning and control in stochastic domains with imperfect information*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[61] David Haussler, *Decision theoretic generalizations of the PAC model*, Information and Computation **100** (1992), 78–150.

[62] ———, *In Wolpert, D. (Ed.): The mathematics of generalization*, vol. XX, ch. Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications, Addison–Wesley, 1995.

[63] Wassily Hoeffding, *Probability inequalities for sums of bounded random variables*, Journal of the American Statistical Association **58** (1963), no. 301, 13–30.

[64] Dean F. Hougen, Maria Gini, and James Slagle, *An integrated connectionist approach to reinforcement learning for robotic control*, Proceedings of the Seventeenth International Conf. on Machine Learning, Morgan Kaufmann, 2000.

[65] Ronald A. Howard, *Dynamic programming and Markov processes*, The MIT Press, Cambridge, MA, 1960.

[66] Junling Hu and Michael P. Wellman, *Multiagent reinforcement learning: Theoretical framework and an algorithm*, Proceedings of the Fifteenth International Conf. on Machine Learning, Morgan Kaufmann, 1998, pp. 242–250.

[67] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan, *Reinforcement learning algorithm for partially observable Markov problems*, Advances in Neural Information Processing Systems, vol. 7, The MIT Press, Cambridge, MA, 1994.

[68] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra, *Planning and acting in partially observable stochastic domains*, Artificial Intelligence **101** (1998), 1–45.

[69] H. Kahn and A. Marshall, *Methods of reducing sample size in Monte Carlo computations*, Journal of the Operations Research Society of America **1** (1953), 263–278.

[70] Sham Kakade, *A natural policy gradient*, Advances in Neural Information Processing Systems, vol. 13, The MIT Press, 2001.

[71] Sham Kakade and Peter Dayan, *Dopamine bonuses*, Neural Networks (2001), 131–137.

[72] Boris Katz, *Using English for indexing and retrieving*, Artificial Intelligence at MIT: Expanding Frontiers, vol. 1, The MIT Press, Cambridge, MA, 1990.

[73] Michael Kearns, Yishay Mansour, and Andrew Y. Ng, *Approximate planning in large POMDPs via reusable trajectories*, Advances in Neural Information Processing Systems, vol. 12, The MIT Press, 1999.

[74] Hajime Kimura and Shigenobu Kobayashi, *An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions*, Proceedings of the Fifteenth International Conf. on Machine Learning, Morgan Kaufmann, 1998, pp. 278–286.

[75] ______, *Reinforcement learning using stochastic gradient algorithm and its application to robots*, Tech. Report 8, The Transaction of the Institute of Electrical Enginners of Japan, 1999.

[76] S. Koenig and R.G. Simmons, *The effect of representation and knowledge on goal-directed exploration with reinforcement learning alogrithms*, Machine Learning **22** (1996), 237–285.

[77] Andrey N. Kolmogorov, *Bounds for the minimal number of elements of an ϵ-net in various classes of functions and their applications to the questions of representability of functions of several variables by superposition of functions of fewer variables*, Uspekhi Mat. Nauk **10** (1955), no. 1, 192–194, (in Russian).

[78] Andrey N. Kolmogorov and V. M. Tikhomirov, *ϵ-entropy and ϵ-capacity of sets in function spaces*, Amer. Math. Soc. Transls. **17** (1961), 277–364, (Uspekhi Mat. Nauk, v.14, vyp. 2, 3-86).

[79] Vijay R. Konda and John N. Tsitsiklis, *Actor-critic algorithms*, Advances in Neural Information Processing Systems, vol. 12, The MIT Press, 1999.

[80] ______, *Actor-critic algorithms*, SIAM Journal on Control and Optimization (2001), 1008–14.

[81] Shailesh Kumar and Risto Miikkulainen, *Confidence-based Q-routing: An on-line adaptive network routing algorithm*, Proceedings of Artificial Neural Networks in Engineering, 1998.

[82] ———, *Confidence based dual reinforcement Q-Routing: An adaptive online network routing algorithm*, Proceedings of the Sixteenth International Joint Conf. on Artificial Intelligence, Morgan Kaufmann, 1999, pp. 758–763.

[83] Daniel D. Lee and H. Sebastian Seung, *Algorithms for non-negative matrix factorization*, Advances in Neural Information Processing Systems, vol. 13, The MIT Press, 2000.

[84] ———, *The manifold ways of perception*, Science **290** (2000), 2268–69.

[85] Nick Littlestone, *Learning quickly when irrelevant attributes abound: A new linear threshold algorithm*, Machine Learning **2** (1988), no. 4, 245–318.

[86] Michael L. Littman, *Markov games as a framework for multi-agent reinforcement learning*, Proceedings of the Eleventh International Conf. on Machine Learning, Morgan Kaufmann, 1994, pp. 157–163.

[87] ———, *Memoryless policies: Theoretical limitations and practical results*, From Animals to Animats 3 (Brighton, UK), 1994.

[88] ———, *Algorithms for sequential decision making*, Ph.D. thesis, Brown University, Providence, RI, 1996.

[89] John Loch and Satinder P. Singh, *Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes*, Proceedings of the Fifteenth International Conf. on Machine Learning, Morgan Kaufmann, 1998.

[90] Gerry E. Loeb, *Prosthetics*, Handbook of Brain Theory and Neural Networks (Michael Arbib, ed.), The MIT Press, 2 ed., 2001.

[91] Yishay Mansour, *Reinforcement learning and mistake bounded algorithms*, Proceedings of the Twelfth Annual Conf. on Computational Learning Theory (New York, NY), ACM Press, 1999, pp. 183–192.

[92] Peter Marbach, *Simulation-based methods for Markov decision processes*, Ph.D. thesis, MIT, Cambridge, MA, 1998.

[93] Peter Marbach, O. Mihatsch, M. Schulte, and John N. Tsitsiklis, *Reinforcement learning for call admission control and routing in integrated service networks*, Advances in Neural Information Processing Systems, vol. 11, The MIT Press, 1998.

[94] Peter Marbach, O. Mihatsch, and John N. Tsitsiklis, *Call admission control and routing in integrated service networks using neuro-dynamic programming*, IEEE Journal on Selected Areas in Communications **18** (2000), no. 2, 197–208.

[95] Peter Marbach and John N. Tsitsiklis, *Simulation-based optimization of Markov reward processes*, Tech. report, MIT, Cambridge, MA, 1998.

[96] ———, *Gradient-based optimization of Markov reward processes: Practical variants*, Machine Learning (2000), 1–25.

[97] Mario Martín, *Reinforcement learning for embedded agents facing complex tasks*, Ph.D. thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 1998.

[98] Andrew K. McCallum, *Reinforcement learning with selective perception and hidden state*, Ph.D. thesis, University of Rochester, Rochester, NY, 1996.

[99] C. McDiarmid, *Surveys in combinatorics*, ch. On the method of bounded differences, pp. 148–188, Cambridge University Press, 1989.

[100] N. Merhav and M. Feder, *Universal prediction*, IEEE Transactions on Information Theory **44** (1998), no. 6, 2124–47, (Invited paper).

[101] Nicolas Meuleau and Paul Bourgine, *Exploration of multi-state environments: Local measures and back-propagation of uncertainty*, Machine Learning **35** (1999), 117–154.

[102] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier, *Solving very large weakly coupled Markov decision processes*, Proceedings of the Fifteenth National Conf. on Artificial Intelligence (Madison, WI), AAAI Press, 1998.

[103] Nicolas Meuleau, Leonid Peshkin, and Kee-Eung Kim, *Exploration in gradient-based reinforcement learning*, Tech. Report 1713, MIT AI lab, Cambridge, MA, 2000.

[104] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie P. Kaelbling, *Learning finite-state controllers for partially observable environments*, Proceedings of the Fifteenth Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1999, pp. 427–436.

[105] Marvin L. Minsky, *Society of mind*, The MIT Press, Cambridge, MA, 1996.

[106] P.R. Montague, P. Dayan, and T.K. Sejnowski, *A framework for mesencephalic dopamine systems based on predictive Hebbian learning*, Journal of Neuroscience **16** (1996), 1936–47.

[107] David E. Moriarty and Risto Miikkulainen, *Efficient reinforcement learning through symbolic evolution*, Machine Learning **22** (1996), 11–32.

[108] ______, *Forming neural networks through efficient and adaptive coevolution*, Evolutionary Computation **5** (1997), no. 4, 373–399.

[109] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette, *Evolutionary Algorithms for Reinforcement Learning*, Journal of AI Research **11** (1999), 199–229.

[110] Kumpati S. Narendra and Mandayam A.L. Thathachar, *Learning automata*, Prentice Hall, 1989.

[111] Andrew Y. Ng and Michael I. Jordan, *PEGASUS: A policy search method for large MDPs and POMDPs*, Proceedings of the Sixteenth Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 2000.

[112] S. Niemi and E. Nummelin, *Central limit theorems for Markov random walks*, Societas Scientiarum Fennica, Helsinki, 1982.

[113] M. Oh and J. Berger, *Adaptive importance sampling in Monte Carlo integration*, Journal of Statistical Computing and Simulation. **41** (1992), 143–168.

[114] Manfred Opper and David Haussler, *Worst case prediction over sequences in under log loss*, The Mathematics of Information Coding, Extraction, and Distribution, Springer Verlag, 1997.

[115] Dirk Ormoneit and Peter W. Glynn, *Kernel-based reinforcement learning in average-cost problems: An application to optimal portfolio choice*, Advances in Neural Information Processing Systems, The MIT Press, 2000.

[116] Luis Ortiz, *Selecting approximately-optimal actions in complex structural domains*, Ph.D. thesis, Brown University, Providence, RI, 2002.

[117] Luis Ortiz and Leslie P. Kaelbling, *Adaptive importance sampling for estimation in structured domains*, Proceedings of the Sixteenth Conf. on Uncertainty in Artificial Intelligence (San Francisco, CA), Morgan Kaufmann, 2000, pp. 446–454.

[118] Martin J. Osborne and Ariel Rubinstein, *A course in game theory*, The MIT Press, 1994.

[119] Christos H. Papadimitriou and John N. Tsitsiklis, *The complexity of Markov decision processes*, Mathematics of Operations Research **12** (1987), no. 3, 441–450.

[120] Leonid Peshkin, *Statistical analysis of brain imaging data*, Master's thesis, Weizmann Institute of Science, Rehovot, Israel, Aug 1995.

[121] ———, *Reinforcement learning by policy search*, Ph.D. thesis, Brown University, Providence, RI, 2001.

[122] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie P. Kaelbling, *Learning to cooperate via policy search*, Proceedings of the Sixteenth Conf. on Uncertainty in Artificial Intelligence (San Francisco, CA), Morgan Kaufmann, 2000, pp. 307–314.

[123] Leonid Peshkin, Nicolas Meuleau, and Leslie P. Kaelbling, *Learning policies with external memory*, Proceedings of the Sixteenth International Conf. on Machine Learning (I. Bratko and S. Dzeroski, eds.), Morgan Kaufmann, 1999, pp. 307–314.

[124] Leonid Peshkin and Sayan Mukherjee, *Bounds on sample size for policy evaluation in Markov environments*, Proceedings of the Fourteenth Annual Conf. on Computational Learning Theory, 2001, pp. 608–15.

[125] Leonid Peshkin and Virginia Savova, *On biological plausibility of policy search*, Proceedings of the Sixth International Conf. on Cognitive and Neural Systems (Boston, MA), May 2002.

[126] Leonid Peshkin and Christian R. Shelton, *Learning from scarse experience*, Proceedings of the Nineteenth International Conf. on Machine Learning, 2002.

[127] Daniel Polani and Risto Miikkulainen, *Fast reinforcement learning through eugenic neuro-evolution*, Tech. Report AI99-277, University of Texas at Austin, Austin, TX, 24 1999.

[128] David Pollard, *Convergence of stochastic processes*, Springer, 1984.

[129] M. Powell, *Direct search algorithms for optimization calculations*, Acta Numerica (1998), 287–336.

[130] M. Powell and J. Swann, *Weighted uniform sampling - a Monte Carlo technique for reducing variance*, Journal of the Institute of Mathematics and Applications **2** (1966), 228–236.

[131] Doina Precup, Richard S. Sutton, and Satinder P. Singh, *Eligibility traces for off-policy policy evaluation*, Proceedings of the Seventeenth International Conf. on Machine Learning, Morgan Kaufmann, 2000.

[132] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical recipes in C: The art of scientific computing*, Cambridge University Press, 1993.

[133] M.L. Puterman, *Markov decision processes*, John Wiley & Sons, 1994.

[134] L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), no. 2, 257–286.

[135] L. R. Rabiner and B. H. Juang, *An introduction to hidden Markov models*, IEEE ASSP Magazine (1986), 4–15.

[136] Mark B. Ring, *Continual learning in reinforcement environments*, Ph.D. thesis, University of Texas at Austin, Austin, TX, 1994.

[137] Benjamin Van Roy, *Learning and value function approximation in complex decision processes*, Ph.D. thesis, MIT, Cambridge, MA, 1998.

[138] R.Y. Rubinstein, *Simulation and the Monte Carlo method*, Wiley, New York, NY, 1981.

[139] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, Parallel Distributed Processing (David E. Rumelhart and James L. McClelland, eds.), vol. 1, The MIT Press, Cambridge, MA, 1986.

[140] D. E. Rumelhart and D. Zipser, *Feature discovery by competetive learning*, Parallel Distributed Processing (David E. Rumelhart and James L. McClelland, eds.), vol. 1, The MIT Press, Cambridge, MA, 1986, pp. 151–193.

[141] Virginia Savova and Leonid Peshkin, *How to compare common and rare words*, Proceedings of the Fifth International Conf. on Cognitive and Neural Systems (Boston, MA), 2001.

[142] Juergen Schmidhuber, *Learning factorial codes by predictability minimization*, Neural Computation **4** (1992), no. 6, 863–879.

[143] ———, *On learning how to learn learning strategies*, Tech. Report FKI-198-94, IDSIA, Lugano, Swiss, 1994.

[144] Juergen Schmidhuber, M. Eldracher, and B. Foltin, *Semilinear predictability minimzation produces well-known feature detectors*, Neural Computation **8** (1996), no. 4, 773–786.

[145] Juergen Schmidhuber and R. Huber, *Learning to generate artificial fovea trajectories for target detection*, International Journal of Neural Systems **2** (1991), 135–141.

[146] Juergen Schmidhuber and Jieyu Zhao, *Direct policy search and uncertain policy evaluation*, Tech. Report IDSIA-50-98, IDSIA, 1998.

[147] Juergen Schmidhuber, Jieyu Zhao, and Marco Wiering, *Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement*, Machine Learning **28** (1997), 105–130.

[148] Jeff Schneider, Weng-Keen Wong, Andrew W. Moore, and Martin Riedmiller, *Distributed value functions*, Proceedings of the Sixteenth International Conf. on Machine Learning (I. Bratko and S. Dzeroski, eds.), Morgan Kaufmann, 1999, pp. 371–378.

[149] Wolfram Schultz, *Predictive reward signal of dopamine neurons*, Journal of neurophysiology **80** (1998), 1–27.

[150] Wolfram Schultz, Peter Dayan, and P. Montague, *A neural substrate of prediction and reward*, Science **275** (1997), 1593–1599.

[151] Christian R. Shelton, *Importance sampling for reinforcement learning with multiple objectives*, Ph.D. thesis, MIT, Cambridge, MA, 2001.

[152] ______, *Policy improvement for POMDPs using normalized importance sampling*, Proceedings of the Seventeenth Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 2001.

[153] Yuri M. Shtarkov, *Universal sequential coding of single measures*, Problems of Information Transmission (1987), 175–185.

[154] Herbert A. Simon, *The sciences of the artificial*, The MIT Press, Cambridge, MA, 1996.

[155] John Simpson and Edmund Weiner (eds.), *Oxford English dictionary*, second ed., Oxford University Press, Oxford, UK, 1989.

[156] Satinder P. Singh and Dimitri P. Bertsekas, *Reinforcement learning for dynamic channel allocation in cellular telephone systems*, Advances in Neural Information Processing Systems, vol. 10, The MIT Press, 1997, pp. 974–980.

[157] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan, *Learning without state-estimation in partially observable Markovian decision processes*, Proceedings of the Eleventh International Conf. on Machine Learning, Morgan Kaufmann, 1994.

[158] ______, *Model-free reinforcement learning for non-Markovian decision problems*, Proceedings of the Eleventh International Conf. on Machine Learning, Morgan Kaufmann, 1994, pp. 284–292.

[159] Satinder P. Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvari, *Convergence results for single-step on-policy reinforcement-learning algorithms*, Machine Learning **38** (2000), no. 3, 287–290.

[160] Richard D. Smallwood and Edward J. Sondik, *The optimal control of partially observable Markov processes over a finite horizon*, Operations Research **21** (1973), 1071–1088.

[161] Edward J. Sondik, *The optimal control of partially observable Markov processes*, Ph.D. thesis, Stanford University, Stanford, CA, 1971.

[162] ______, *The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs*, Operations Research **26** (1978), no. 2, 282–304.

[163] Jerome Spanier, *A new family of estimators for random walk problems*, Journal of the Institute of Mathematics and Applications **23** (1979), 1–31.

[164] Malcolm Strens and Andrew W. Moore, *Direct policy search using paired statistical tests*, Proceedings of the Eighteenth International Conf. on Machine Learning, Morgan Kaufmann, 2001.

[165] Devika Subramanian, Peter Druschel, and Johnny Chen, *Ants and reinforcement learning: A case study in routing in dynamic networks*, Proceedings of the Fifteenth International Joint Conf. on Artificial Intelligence, 1997, pp. 832–839.

[166] Richard S. Sutton, *Open theoretical questions in reinforcement learning*, http://www-anw.cs.umass.edu/~rich/publications.html, 1999, manuscript.

[167] Richard S. Sutton and Andrew G. Barto, *Reinforcement learning: An introduction*, The MIT Press, Cambridge, MA, 1998.

[168] Richard S. Sutton, David McAllester, Satinder P. Singh, and Yishay Mansour, *Policy gradient methods for reinforcement learning with function approximation*, Advances in Neural Information Processing Systems, vol. 12, The MIT Press, 1999, pp. 1057–63.

[169] Csaba Szepesvari, *personal communication*, Aug 1999.

[170] Stribel C. T., *Sufficient statistics in the optimal control of stochastic systems*, Journal of Mathematical Analysis and Applications **12** (1965), 576–592.

[171] *TAC - the 2001 trading agent compeitition*, ACM Conference on Electronic Commerce, 10 2001, http://tac.eecs.umich.edu/.

[172] Nigel Tao, Jonathan Baxter, and Lex Weaver, *A multi-agent, policy gradient approach to network routing*, Proceedings of the Eighteenth International Conf. on Machine Learning, Morgan Kaufmann, 2001.

[173] Gerald J. Tesauro, *Practical issues in temporal difference learning*, Machine Learning **8** (1992), 257–277.

[174] Sebastian Thrun, *Efficient exploration in reinforcement learning*, Tech. Report CS-92-102, CMU, Pittsburgh, PA, 1992.

[175] John N. Tsitsiklis, *Asynchronous stochastic approximation and Q-learning*, Machine Learning **16** (1994), no. 3, 185–202.

[176] *http://www.ubique.com/*, 1995, Virtual Places.

[177] Shimon Ullman, *High-level vision*, The MIT Press, Cambridge, MA, 1996.

[178] L. G. Valiant, *A theory of the learnable*, Communications of the ACM **27** (1984), no. 11, 1134–1142.

[179] Vladimir Vapnik, *Statistical learning theory*, Wiley, 1998.

[180] Vladimir Vapnik, Esther Levin, and Yann Le Cun, *Measuring the VC-dimension of a learning machine*, Neural Computation **6** (1994), no. 5, 851–876.

[181] Eric Veach, *Robust monte carlo methods for light transport simulation*, Ph.D. thesis, Stanford University, Palo Alto, CA, 1997.

[182] Johann Wolfgang von Goethe, *Faust*, vol. 2, 1833.

[183] Chris J. C. H. Watkins, *Learning from delayed rewards*, Ph.D. thesis, King's College, Cambridge, UK, 1989.

[184] Chris J. C. H. Watkins and Peter Dayan, *Q-learning*, Machine Learning **8** (1992), no. 3, 279–292.

[185] Marco Wiering and Juergen Schmidhuber, *HQ-learning*, Adaptive Behavior **6** (1998), no. 2, 219–246.

[186] R. Williams and J. Peng, *Function optimization using connectionist reinforcement learning algorithms*, Connection Science **3** (1991), 241–268.

[187] Ronald J. Williams, *Reinforcement learning in connectionist networks: A mathematical analysis*, Tech. Report ICS-8605, Institute for Cognitive Science, University of California, San Diego, La Jolla, CA, 1986.

[188] ______, *A class of gradient-estimating algorithms for reinforcement learning in neural networks*, Proceedings of the IEEE First International Conference on Neural Networks (San Diego, CA), 1987.

[189] ______, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine Learning **8** (1992), no. 3, 229–256.

[190] David H. Wolpert, Kagan Tumer, and Jeremy Frank, *Using collective intelligence to route internet traffic*, Advances in Neural Information Processing Systems, vol. 11, The MIT Press, 1998, pp. 952–8.

[191] Wei Zhang and Thomas G. Dietterich, *Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling*, Journal of AI Research **1** (2000), 1–38.

[192] Y. Zhou, *Adaptive importance sampling for integration*, Ph.D. thesis, Stanford University, Palo Alto, CA, 1998.

[193] M. Zlochin and Y. Baram, *The bias-variance dilemma of the Monte Carlo method*, http://iridia.ulb.ac.be/~mzlochin, 2000, unpublished.

Notation

A	the set of actions (p. 7)
$a(t)$	an action at time t (p. 7)
B	the observation function (p. 7)
$\underline{c}$	the minimal probability of action in stochastic policy (p. 11)
D	the data set (p. 84)
E	the exploration trace (p. 24)
G	a collection of agents (p. 44)
H	a set of experiences (p. 10)
h	the experience (p. 10)
L	a regret in sequential prediction game (p. 103)
M	a finite set of internal controller states (p. 34)
N	counter of events (p. 23) and index size
$\mathcal{N}$	the covering number (p. 95)
O	the set of observations (p. 7)
$R(h)$	the return of the experience h (p. 10)
$r(t)$	a feedback at time t (p. 7)
S	the set of states (p. 7)
$\mathcal{T}$	the state transition function (p. 7)
t, τ	a time step (p. 7)
$V(\theta)$	a value of the policy θ (p. 12)
w	a likelihood ratio
x, y	cart and pole positions in a pole balancing domain (p. 39)
α	a learning rate
β	bias of an estimator
γ	the discount factor
Δ	an update for policy parameter (p. 22)
δ	a confidence value
ϵ	a small number
η	the upper bound on likelihood ratio (p. 101)
Θ	the space of policies (p. 11)
θ	the parameterization of a policy (p. 11)
κ	a register variable for likelihood ratio
λ	an exploration ratio (p. 87)
μ, μ_d	a policy function (p. 11)
ρ	the feedback function (p. 7)
σ	standard deviation
ζ	the temperature in Boltzmann process (p. 25)
$\Phi(h)$	the factor in the probability of the experience h related to the environment (p. 22)
Ψ	the factor in the probability of the experience h related to the agent (p. 22)

Index