Learning Parameterized Skills

Bruno Castro da Silva
University of Massachusetts - Amherst

LEARNING PARAMETERIZED SKILLS

A Dissertation Presented

by

BRUNO CASTRO DA SILVA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2015

Computer Science

LEARNING PARAMETERIZED SKILLS

A Dissertation Presented

by

BRUNO CASTRO DA SILVA

for my family

ACKNOWLEDGMENTS

I first read Andrew Barto's book on reinforcement learning in 2005. I remember being thoroughly amazed by the elegance and succinctness with which he introduced the field that he and Rich Sutton pioneered. It was difficult to imagine that two years later I would be living in a different country and working with Andy himself. Andy's breadth of knowledge in math, psychology, statistics, neuroscience, and machine learning make him a perfect example of a Renaissance man. His ability to see the underlying themes connecting seemingly disparate ideas is uncanny. Andy's technical skills are only surpassed by his immense generosity, thoughtfulness, and completely unnecessary humbleness. I cannot imagine a better role model to aspire to. Thank you, Andy, for these incredible seven years.

I am also grateful to Victor Lesser. Victor's guidance began even before I applied to this program and has remained invaluable ever since. His continuous support allowed me to pursue my interests even when they did not perfectly align with his. I am glad that I got the chance to work with him. I am also thankful for Sridhar Mahadevan's sound and practical advice. Sridhar's ideas were essential to my research, and in fact part of this thesis directly follows from his earlier studies on manifold learning. Rod Grupen and Neil Berthier—the other members of my thesis committee—provided early feedback that made this dissertation possible. I am also indebted to Gianluca Baldassarre. Gianluca's creativity and ability to ask the important questions helped me consolidate the idea of parameterized skills. His support, dedication, and friendship were invaluable during my time in Rome and were utterly crucial to our efforts to achieve skill learning on a robot.

Many of the ideas in this thesis originate from discussions with my colleagues in the Autonomous Learning Lab. George Konidaris, in particular, was a constant source of inspiration and encouragement. His insights and intellectual curiosity underlie many of my papers, and his unrelenting optimism made all of this possible. Thanks, dude! Many others have contributed to making my journey through grad school fun: Phil Thomas, Bo Liu, Yariv Levy, Will Dabney, Scott Kuindersma, Armita Kaboli, Vimal Mathew, Scott Niekum, Livia Quintanilha, Ashvin Shah, Özgür Şimşek, Chris Vigorito, Andrew Stout, Chang Wang, Yoonheui Kim, and Amir Tal. I am also grateful to the wonderful people I met during orientation week and who have remained by my side, one way or another, ever since. Blake Foster, Dirk Ruiken, Julie de Chantal and Biki Takashima-Uebelhör: you made this little town feel like home.

When I first arrived in Amherst, with two broken suitcases and a guitar, I was met by two amazing Brazilians who later became my family away from home. Antonio Americo and Andre Muniz, vocês são meus irmãos. Obrigado pelas risadas, churrascos e rodas de violão. I am also grateful for the support of the extended family I formed over these years: Daniel Sadoc, Jacqueline Muniz, Antonio "Guto" Rocha, Bruno Ribeiro, Paula Franco, Gabi Dantas, Fabricio Murai, Cibele Freire, Marcelo Dias, Paula Condon, Karin Camihort, and Linda Pisano. Many other friends helped from afar. Daniel Basso has been a constant presence in my life ever since we drunkenly discussed Kohonen maps on a bus to the Pantanal, a decade ago. Carlos Scheidegger, Adelio Bilhalva, Enguer Charão, Rafa Perotto, and Thais Santa Rita: you are the friends I wish I could see every day.

I am grateful to my family. Their love and unconditional support were a constant source of motivation and strength. Not a single day goes by when I don't think about how privileged I am. My parents and sisters have been nothing but supportive and encouraging throughout this entire process—even when I decided to move 5000 miles away. I am forever indebted to them for the many sacrifices they have made to give me a life that few in my country have. Mãe e Pai, obrigado.

Most of all, I would like to thank my wife, Ana. Her love, quiet support, and constant reassurance kept me sane and gave me a reason to keep going. I could not have done this without her. Ana, seven years ago you agreed to embark with me on this uncertain journey. I am glad you came along and can't wait to see what comes next. *O mundo começa agora. Apenas começamos.*

ABSTRACT

LEARNING PARAMETERIZED SKILLS

FEBRUARY 2015

BRUNO CASTRO DA SILVA

B.Sc., FEDERAL UNIVERSITY OF RIO GRANDE DO SUL, BRAZIL

M.Sc., FEDERAL UNIVERSITY OF RIO GRANDE DO SUL, BRAZIL

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew G. Barto

One of the defining characteristics of human intelligence is the ability to acquire and refine skills. Skills are behaviors for solving problems and performing tasks that an agent encounters often—sometimes in different contexts and situations—throughout its lifetime. Identifying important problems that recur and retaining their solutions as skills allows an agent to more rapidly solve novel problems by adjusting and combining its existing skills.

One of the motivating principles underlying the idea of skill acquisition is that open-ended learning agents need to solve not a single problem but a range of problems over their lifetimes. In this thesis we introduce a *general framework for learning reusable skills*. Reusable skills are parameterized procedures that—given a description of a problem to be solved or task to be performed—produce appropriate behaviors or policies. They can produce, on-demand, policies for any problems in a given family of related problems, even those with which the agent has never had direct experience. We refer to reusable skills of this type as *parameterized skills.* Parameterized skills are useful because they allow an agent to tackle novel variations of a problem given only a parameterized description of the problem. They can be sequentially and hierarchically combined with other skills and primitive actions to produce progressively more abstract and temporally extended behaviors.

Existing work has shown that it is possible to transfer information between pairs of related problems and that parameterized policies can be constructed to deal with slight variations of a known problem. Not much attention, however, has been given to methods that allow agents to autonomously and actively learn general, reusable parameterized skills from few training examples. In this thesis we identify three major challenges involved in the construction of such skills.

First, *an agent should be capable of solving a small number of problems and generalizing these experiences to construct a single reusable skill.* The skill should be capable of producing—on demand—appropriate behaviors even when applied to yet unseen variations of a problem or task. Once learned, parameterized skills might be sequentially or hierarchically combined with other skills to produce increasingly more general behaviors;

Secondly, *an agent should be able to identify when a parameterized skill can be hierarchically decomposed into specialized sub-skills.* A single skill can often be hierarchically decomposed into specialized sub-skills, each one capable of solving one particular subset of problems. As an example, a parameterized throwing skill might be hierarchically expressed as the composition of specialized overhand and underhand throwing skills. Even though these might involve qualitatively different motor behaviors, they are nonetheless instances of a same overall problem—that of throwing an object—and can be encapsulated into a *single* general skill for throwing. The agent should be capable to analyze the parameters of a particular target location and identify which sub-skill is the most appropriate for hitting it. Identifying and modeling sub-skills allows agents to *autonomously aggregate* related parameterized behaviors into single, more abstract skills;

Finally, *the agent should be able to actively select on which problems it wishes to practice in order to more rapidly become competent in a skill.* Thoughtful and deliberate practice is one of the defining characteristics of human expert performance. By carefully choosing on which problems to practice, and in what order, the agent may be able to more rapidly construct a skill that performs well over a wide range of related problems.

We address these challenges by introducing a general framework for learning parameterized skills. We evaluate it on challenging simulated decision-making problems and on a physical humanoid robot, and we demonstrate that it allows for the efficient and active construction of reusable skills from limited data.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

One of the defining characteristics of human intelligence is the ability to acquire and refine skills. Skills are behaviors for solving problems and performing tasks that an agent encounters often—sometimes in different contexts and situations—throughout its lifetime. Identifying important problems that recur and retaining their solutions as skills allows an agent to more rapidly solve novel problems by adjusting and combining its existing skills.

One of the motivating principles underlying the idea of skill acquisition is that open-ended learning agents need to solve not a single problem (which we interchangeably refer to as a *task*), but rather a range of problems over their lifetimes. *Skills* can be used to encapsulate the behavioral knowledge needed to solve particular types of problems. Once learned, they can be combined with other skills to address progressively more complex challenges. Consider, as an example, the motor skills of reaching and grasping. Humans learn such skills early in their lives [7] and subsequently use them as basic "building blocks" for solving increasingly more general manual manipulation problems. Skill composition of this kind allows an agent to address complex problems by first decomposing them into more tractable subproblems. Learning to fly an airplane, for instance, may be infeasible if treated as a single problem. It can, however, be expressed as a combination of simpler behaviors—for which specialized skills can be learned—such as taxiing, steering and controlling the rudders and pitch of the airplane. Acquiring skills of this type widens the scope of an agent's behavior and allows it to tackle learning problems that would otherwise be infeasible or extremely challenging.

In this thesis we introduce a *general framework for learning reusable skills.* Optimization techniques exist that allow an agent to learn optimal or near-optimal behaviors, or *policies*, for solving single decision-making problems. These policies, however, may fail if the problems at hand vary or if the policies are to be applied in novel contexts. As an example, consider a grasping skill. It should allow an agent to grasp not only a single type of object, but objects of various shapes and weights. It should also be *reusable* in a wide range of contexts; for instance, situations in which single or two-handed grasps may be required. A skill is said to be *reusable* if it consists in a parameterized procedure that—given a description of the problem to be solved—produces an appropriate behavior or policy. In what follows we refer to these procedures as *parameterized skills.*

Parameterized skills are useful for solving families of related problems that recur throughout an agent's lifetime. Once learned they can produce—on-demand—policies for any problems in a given distribution, even those with which the agent has never had direct experience. As an example, consider a soccer-playing agent tasked with learning a kicking skill parameterized by desired force and direction. For it to be truly competent, it should be able to execute such kicks whenever necessary, even with a particular combination of force and direction it has never had direct experience with. Learning a single policy for each possible type of kick is infeasible. The agent might wish, instead, to learn policies for a few specific kicks and use them to synthesize a single general skill for kicking, parameterized by force and direction, that it can execute on-demand.

Parameterized skills can be sequentially and hierarchically combined with other skills and primitive actions to produce progressively more abstract and temporally extended behaviors. Skill composition of this type is important because it allows for the hierarchical organization of an agent's behavior. A key advantage of hierarchical behaviors is that skills, once learned, can be treated as *adjustable primitive actions* and incorporated into higher-level skills, thereby abstracting away details of lower-level control.

Existing work has shown that it is possible to transfer information between pairs of related problems [76, 75, 44, 3] and that parameterized policies can be constructed to deal with slight variations of a domain [59, 53, 33, 27]. Not much attention, however, has been given to methods that allow an agent to autonomously and actively learn *general, parameterized skills* from scratch and with few training samples.

1.1 Research Challenges

In this dissertation we identify three problems that need to be addressed for an agent to autonomously and efficiently learn parameterized skills:

1. *The agent should be capable of solving a small number of problems and generalizing these experiences to construct a single reusable skill.* The skill should be capable of producing—on demand—appropriate behaviors even when applied to yet unseen variations of a problem or task. Once learned, it may be combined with other skills or primitive actions to produce progressively more abstract and temporally extended behaviors;

2. *The agent should be able to identify when a given skill can be hierarchically decomposed into sub-skills specialized in solving different classes of problems.* As an example, consider the kicking skill introduced above. Qualitatively different types of kicks may be needed during a game; for instance, free kicks or kicks to pass the ball to a different player. The choice of which one to use depends on the current objective of the agent. A general parameterized skill for kicking may, therefore, be hierarchically expressed as the composition of these two *sub-skills*,

each of which is specialized in executing one particular type of kick. Even though these might involve qualitatively different motor behaviors, they are nonetheless instances of a same overall problem—that of kicking a ball—and might be encapsulated into a *single* general skill for kicking. The agent should be capable of analyzing the parameters of a particular kick and identifying which sub-skill is the most appropriate for executing it. Identifying and modeling sub-skills allows agents to *autonomously aggregate* related parameterized behaviors into single, more abstract skills. This is important to reduce the number of actions available to an agent and to simplify the space of possible policies;

3. *The agent should be able to autonomously select on which tasks it wishes to practice in order to more rapidly become competent in a skill.* Intuitively, the tasks from which experience is most beneficial are those that allow a skill to better generalize to a wider range of related problems. This goal can be formalized as an active task selection criterion whose objective is to maximize skill performance over a family of related problems.

1.2 Contributions

This dissertation makes three main contributions, each one addressing one of the major challenges introduced above.

1. ***A method to learn reusable parameterized skills***. We introduce a skill-learning method that solves a small number of decision-making problems and uses the corresponding learned policies to estimate properties of the lower-dimensional manifold on which they lie. This manifold models how policy parameters change as we vary the parameters of a problem. Our method combines manifold learning techniques, classification algorithms and non-linear regression methods to estimate geometric and topological properties of manifold.

This allows agents to obtain a general model by which policy parameters can be predicted from task parameters.

2. ***A method for autonomously discovering the sub-skills that may be needed to solve different classes of problems***. We introduce a method for automatically identifying and modeling the qualitatively different and specialized sub-skills that might be needed to implement a given skill. We observe that the manifold where task policies lie may be composed of several disjoint, piecewise-smooth surfaces, or charts. Disjoint charts might exist because changes in task or problem parameters may result in abrupt changes in the parameterization of its policy. Each disjoint chart typically encodes a specialized *sub-skill* needed to solve a particular subclass of problems. We propose a method for identifying and modeling each of these specialized sub-skills, thereby obtaining a unified model by which different sub-skills are integrated into a single, more general parameterized behavior. Furthermore, *we introduce a rejection sampling method designed to make skill-learning more sample-efficient.* This method is motivated by the observation that unsuccessful policies experienced while learning to solve one particular problem may be useful to solve different, but possibly related, problems. Realizing when this is the case corresponds to a type of counterfactual reasoning that allows agents to use seemingly unsuccessful policies as additional valid training samples to construct a skill, thus accelerating the skill acquisition process.

3. ***A method to actively select on which tasks to practice in order to more rapidly become competent in a skill***. We introduce a method for actively learning parameterized skills. We present a non-parametric Bayesian model of skill performance and derive analytical expressions for a novel acquisition criterion capable of identifying tasks that maximize expected improvement

in skill performance. We also introduce a spatiotemporal kernel specially tailored for non-stationary skill performance models. Our active learning framework is agnostic to policy and skill representation and scales independently of task dimension.

We evaluate these methods on challenging simulated decision-making problems and on a physical humanoid robot tasked with learning whole-body parameterized motor skills from limited data.

This thesis is organized as follows. In Chapter 2 we introduce the background necessary to understand the remainder of this dissertation and discuss relevant existing research. In Chapter 3 we introduce a method for learning parameterized skills from a small number of training examples. We extend this method in Chapter 4 so that it is applicable to robotics problems, where samples are expensive to collect. We discuss how to automatically identify and model the qualitatively different sub-skills needed to implement a skill, and we introduce a rejection sampling method designed to make skill-learning more sample-efficient. Chapter 5 focuses on how an agent might actively select how to practice in order to more rapidly become competent in a skill. Finally, we present possible future work in Chapter 7 and conclude with a discussion and summary of the main contributions of this thesis.

CHAPTER 2
BACKGROUND AND RELATED WORK

This chapter presents a brief overview of Reinforcement Learning and describes existing work relevant to the high-level objective of skill acquisition. Research more specifically related to each of the individual algorithms developed in this dissertation is discussed in the relevant chapter.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a framework for learning how to act in complex environments while maximizing a reward signal [74]. An RL agent interacts with its environment in discrete time steps. At each time, it observes the current state of the environment and selects an action. After executing an action, the agent observes its effects—the new state of the environment—and receives a reward signal. The behavior of an agent is encoded by a *policy*—a function specifying which actions should be taken in different situations. The objective of the agent is to learn a behavior, or policy, that allows it to maximize the cumulative reward it receives over time.

RL problems are typically modeled as Markov Decision Processes (MDP). An MDP M is a tuple $\big(S, A, R, T, \gamma\big)$, where S is a finite set of states in which the agent may find itself; A is a finite set of actions that the agent may choose to execute; $R : S \times A \to \mathbb{R}$ is a function returning a scalar reward signal for executing action $a \in A$ in state $s \in S$; $T : S \times A \times S \to [0, 1]$ is a transition function specifying the probability $P(s'|a, s)$ of the agent transitioning to state $s' \in S$ after taking action a

while in state s; and $\gamma \in [0,1)$ is a discount factor expressing the extent to which the agent prefers immediate over delayed rewards. Finally, a policy $\pi : S \times A \rightarrow [0,1]$ is a function that encodes how the agent should behave (i.e., which action it should take in each state). Concretely, $\pi(s,a)$ is the probability $P(a|s)$ of selecting action a when in state s.

The goal of an RL agent is to learn a policy that allows it to accumulate as much reward as possible. Let the reward received at time t be r_t and the cumulative reward (or *return*) from time t on be $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$. Solving an MDP M consists of finding an optimal policy π^* that maximizes the agent's expected return. Let $V^\pi(s)$ be the expected cumulative discounted reward obtained when starting in state s and following policy π:

$$\begin{aligned} V^\pi(s) &= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s\right] \\ &= \sum_a \pi(s,a)\left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V^\pi(s')\right] \end{aligned}$$

where E_π denotes the expected value with respect to a policy π and t is any time step. This function is called a *value function.* Similarly, let $Q^\pi(s,a)$ be the expected total discounted reward obtained when taking action a in state s and following π thereafter:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\left[\sum_{k=0}^{\infty} \gamma^t r_{t+k} \,\middle|\, s_t = s,\, a_t = a\right] \\ &= R(s,a) + \gamma \sum_{s'} T(s,a,s') \sum_{a'} \pi(s',a') Q^\pi(s',a') \end{aligned}$$

The optimal Q-function for an MDP M, denoted by Q^*, is the function obtained when following an optimal policy: $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. It has the property that

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^*(s',a') \tag{2.1}$$

One way of identifying optimal policies is to first compute or estimate $Q^*(s,a)$ and then select greedy actions with respect to it. In this case, an optimal policy is defined as $\pi^*(s) = \arg\max_a Q^*(s,a)$. Optimal policies can also be computed via dynamic programming techniques if T and R are known and the number of states in S is not large. If models are not available or if the state space is too large, exact methods may become infeasible and approximation techniques are needed. These techniques are generally referred to as *approximate dynamic programming.*

RL methods often learn optimal policies via *temporal difference* (TD) techniques [74], which iteratively improve estimates of V^* or Q^*. One commonly-used TD control method is Q-Learning [83]. Let Q^t be an agent's estimate of Q^* at time t. A Q-Learning agent in state s_t selects an action a_t according to some *arbitrary* policy and then observes a subsequent reward r and a next state s'. Based on these, it updates its estimate Q^t according to

$$Q^{t+1}(s_t,a_t) \leftarrow Q^t(s_t,a_t) + \alpha_t \Big(r + \gamma \max_a Q^t(s_{t+1},a) - Q^t(s_t,a_t) \Big) \tag{2.2}$$

where $\alpha_t \in (0,1]$ is a step-size parameter. Note that Equation 2.2 does not assume knowledge of the transition or reward functions. For this reason, Q-learning is known as a *model-free* method. Under mild assumptions regarding the sequence of step sizes used, and assuming that all actions are executed infinitely often in each state, the sequence of estimates Q^t provably converges to Q^* [74].

2.1.1 Value Function Approximation

In several practical problems, states of an MDP are not just arbitrary symbols with no internal structure. They can often be defined as tuples of real-valued *features.* In a robot application, for instance, a state may correspond to a vector that aggregates the current angle and angular velocities of each one of the robot's joints. This type of state representation has two important consequences: *(1)* value functions now have to be defined over continuous state spaces; *(2)* during an agent's interaction with the environment, individual states might be visited only once—if at all—and therefore a mechanism for generalizing experiences from nearby states becomes necessary.

One common way of compactly representing continuous value functions, while generalizing values to nearby states, is via a linear combination of basis functions $\mathbf{\Phi}$, each of which depends on the state:

$$\hat{V}(s) = \mathbf{w} \cdot \mathbf{\Phi}(s) = \sum_{i=1}^{n} w_i \phi_i(s). \tag{2.3}$$

Note that even though $\hat{V}(s)$ is a linear model, it can represent complex values functions because the bases ϕ_i themselves are allowed to be arbitrarily complex. Popular choices for basis functions include Radial Basis Functions and the Fourier Basis [39]. Approximating a value function, under this setting, corresponds to identifying an appropriate weight vector $\mathbf{w}$. Temporal difference errors may be combined with a gradient descent step to update the value function's weights at time t:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\Bigg(r + \gamma V(s') - V(s)\Bigg)\nabla_{\mathbf{w}_t} V(s) \\ &= \mathbf{w}_t + \alpha\Bigg(r + \gamma V(s') - V(s)\Bigg)\mathbf{\Phi}(s).\end{aligned}$$

2.1.2 Direct Policy Search

In some cases, complex value functions may induce simple policies. In these situations it may be easier to directly search the space of parameterized policies for policies that maximize expected return, instead of first estimating an optimal value function. Policy search methods exploit this idea. They assume that policies are parameterized functions that directly map states to actions or to probability distributions over actions. By modifying the parameters of a policy, different behaviors can be obtained.

Let us assume that policies are functions π_θ parameterized by vectors $\theta \in \mathbb{R}^M$ of weights. If the policy is stochastic the weights θ directly parameterize a probability distribution over actions. Alternatively, π_θ may be *independent* of the current state (i.e., *open-loop*), in which case θ parameterizes fixed trajectories through the state space.

The goal of a policy search method is to directly identify policy parameters that induce an optimal or near-optimal policy. This type of search may have advantages over value function-based techniques. First, they might allow agents to more naturally deal with continuous states and actions. Secondly, policy representations can be chosen that are meaningful for the task at hand and that easily incorporate domain knowledge. Thirdly, since parameterized policies often require fewer parameters than value functions, they may be easier to estimate. Fourthly, direct policy search methods often do not depend on the Markov property—the requirement that the conditional probability distribution over future states depends *only* on the current state. Finally, some policy search methods, such as policy gradient techniques, are known to converge to local optima and can be used in a model-free manner, or, alternatively, can make use of a model if one is available.

Direct policy search can be performed in three main ways: *(1)* by using black-box local search methods, such as hill climbing or the NelderMead simplex technique [34]; *(2)* by using gradient descent methods [86, 6, 72, 9]; or *(3)* by using EM or cross-entropy approaches [31, 46]. In some cases, EM and cross entropy methods can be shown to be equivalent and correspond to special cases of a broader class of learning techniques based on probability-weighted averages of policy parameters [70].

Policy gradient methods improve a policy by updating its parameters along the gradient of the expected return $J(\theta)$ with respect to policy parameters θ. Most applications of policy gradient algorithms to robotics involve *undiscounted, episodic and indefinite-horizon tasks*, in which case $J(\theta)$ is simply referred to as *policy performance*. **In the following chapters we use J instead of V whenever we wish to clarify that the problem being considered is undiscounted ($\gamma = 1$), episodic and with indefinite-horizon**.

Let $\pi_\theta(s, a)$ be a differentiable policy (with respect to its parameters θ) defining the probability of selecting action a when in state s. The objective of a policy gradient algorithm is to identify the parameters θ^* that maximize the expected return of the policy:

$$\begin{aligned} \theta^* &\equiv \arg\max_\theta E_{\mathbb{T}}\Big[J(\theta)\Big] \\ &= \arg\max_\theta E_{\mathbb{T}}\Big[R(s_0, a_0) + R(s_1, a_1) + \ldots\Big] \end{aligned}$$

where the expectation is over all possible trajectories $\tau \in \mathbb{T}$ resulting from the execution of policy π_θ. It is possible to write the gradient of $J(\theta)$, with respect to the parameters θ of a policy, as:

$$\nabla_\theta J(\theta) = E_{\mathbb{T}}\left[R(\tau)\left(\frac{\nabla_\theta \pi_\theta(s_0, a_0)}{\pi_\theta(s_0, a_0)} + \ldots + \frac{\nabla_\theta \pi_\theta(s_T, a_T)}{\pi_\theta(s_T, a_T)}\right)\right]$$

where the expectation is over all possible trajectories $\tau \in \mathbb{T}$ resulting from the execution of policy π_θ and T and $R(\tau)$ are, respectively, the length and the return of trajectory τ. The gradient $\nabla_\theta J(\theta)$ is the direction in θ-space of most rapid increase in policy performance J. It may be approximated in several ways—for instance, by sampling trajectories τ of π_θ. Such a gradient, once estimated, may be used to update the parameters of a policy via a gradient ascent approach:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta$$

where α is a step-size parameter. This specific gradient-based policy update is called REINFORCE [86]. If the Q-function Q^π under the current policy π_θ is known, it can be shown [72] that

$$\nabla_\theta J(\theta) = \sum_s d^\pi(s) \sum_a \nabla_\theta \pi_\theta(s,a) Q^\pi(s,a),$$

where $d^\pi(s) = lim_{t\to\infty} Pr(s_t = s|s_0, \pi)$ is the stationary distribution of states under π, which we assume exists and is independent of s_0 for all policies. More advanced gradient methods exist, such as those based on natural gradients [9] and mirror descent algorithms [45], which often allow for faster convergence to near-optimal policies.

2.2 Related Work

We now present existing related work relevant to the problem of learning parameterized skills.

2.2.1 The Options Framework

The standard MDP setting assumes that actions are atomic operations that take exactly one time step to be executed. When a particular control problem is modeled as an MDP, its actions are assumed to correspond to primitive behaviors, pre-defined and handcrafted by a designer. Reinforcement learning agents are typically concerned with learning to *select* actions, but not necessarily to *construct* new actions by combining existing ones, or to identify useful behaviors that can be learned and stored for later reuse.

One way of addressing these limitations is via the *options framework* [73]. The Options framework describes to a set of formalisms and methods for dealing with hierarchical Reinforcement Learning (HRL) problems and for learning and planning using temporally extended actions. Temporally extended actions, also known as options, are named, reusable policies defined in terms of primitive actions or other options.

One of the motivating principles underlying this idea is that subproblems recur, so that options can be reused in a variety of related tasks or contexts. In particular, options are useful because they abstract away details of low-level control and allow for the definition of hierarchical policies.

An option o consists of three components: *(1)* a policy $\pi_o(s, a)$ describing the probability of taking action a while executing option o in state s; *(2)* an initiation set I_o containing all states $s \in S$ in which the option can be executed; and *(3)* a termination condition β_o giving the probability of the option terminating at each state in which it is defined. Option policies are typically defined over subsets of the state space in which they are embedded, though this is not always required. Konidaris and Barto [37], for example, proposed a method for identifying the best state abstractions to be used by each option.

Because MDPs do not allow for the use of temporally extended actions, a more flexible formalism is needed. One way in which MDPs can be extended is via the Semi-Markov Decision Processes (SMDP) framework [54]. In an SMDP, $T(s', \tau|s, o)$ is the probability of reaching state s' after τ time steps when executing option o from state s, and $R(s, o)$ describes the total discounted return accumulated during the execution of o. Option policies can then be learned using standard RL algorithms.

Options are typically handcrafted by a designer who defines their objectives via an option reward function R_o. An option for walking through a door, for instance, might reward states in which the agent reaches or approaches that door. Algorithms for *autonomously constructing* options also exist. These include methods for *(1)* determining when to create a new option; *(2)* autonomously identifying a suitable option reward function; *(3)* identifying what should the initiation and termination conditions be; and *(4)* determining whether any of the conditions or metaparameters defining an option should to be dynamically modified with time. Initiation and termination conditions, for instance, might be determined by identifying possibly useful subgoal states [47, 14, 16, 38] that an agent may want to reach.

The Options framework is an important first step towards an architecture and set of algorithms capable of specifying and learning reusable skills. Unfortunately, options only allow for a *single* policy—specifically, the one maximizing the option reward function R_o. They are not flexible enough to encode arbitrary reusable skills capable of solving novel variations of a problem, since each variation may require a different, specialized policy. In Chapter 3 we introduce a type of *parameterized option* that produces appropriate behaviors or policies based only on a description of the problem to be solved. We argue that this method extends the Options framework in a way that achieves one of its original motivating objectives: the autonomous acquisition of higher-level reusable skills from data.

2.2.2 Parameterized Options via State Space Augmentation

Consider a robot tasked with delivering mail to different offices in a building. This robot may benefit from a set of options, $o_1, \ldots, o_n$, each one specifying a policy for reaching one of the n rooms in the building. Assume that these policies are defined over some state space S_o. Instead of explicitly learning and storing n separate options, the robot may prefer to construct a *single* parameterized option, capable of reaching any room in the building given only the number of that room. Parameterized options of this type may be constructed by posing the problem of learning to reach n rooms as a single MDP, whose state space S_{aug} augments S_o by including the target room's number as an additional state feature. Even though this is possible in principle, we will argue, in what follows, that this approach has important limitations. The augmented state space S_{aug} of the problem introduced above is defined as $\{(\textit{target-room}, s_o) \mid \textit{target-room} \in \{1, \ldots, n\} \text{ and } s_o \in S_o\}$. Assume that rewards in this MDP are given whenever the agent enters a room whose number corresponds to the current value of the *target-room* feature. Once learned, optimal policies π_{aug} for this MDP can be seen as parameterized options. Concretely, an agent wishing to reach a given room k selects, at each time t, actions $a(t)$ with probability $\pi_{\text{aug}}((k, s_o(t)), a(t))$, where $s_o(t)$ is the state of the agent at time t. Intuitively, π_{aug} jointly solves a *set* of decision-making problems—namely, those encoded by the options $o_1, \ldots, o_n$. The particular task solution to be produced by π_{aug} is specified via the additional *target-room* state feature describing the parameters of the task of interest. This approach has important shortcomings:

1. Learning a single policy π_{aug} over n tasks may require a larger number of samples than learning n independent policies, especially if the agent has to estimate a value function. This is because such a value function is defined over a larger state space S_{aug};

2. Learning a single policy over n tasks may require more samples than learning n independent policies if the agent practices one task at a time, and if different tasks may require significantly different policies. Consider an agent that selects one task uniformly at random and practices it until its joint policy π_{aug} can no longer be improved, and then repeats this process over several tasks. Such an agent may have to collect a large number of samples if its current policy is significantly different from one that is appropriate to the new training task. Intuitively, the agent may need to repeatedly "unlearn" its current policy (which might have overfit to the last task) in order to adapt it to subsequent training problems. This type of negative transfer may result in poorer sample complexity[1];

[1]As a simple example, consider an agent faced with n decision-making problems. Assume that these can be grouped into G clusters, each containing problems whose solutions are similar according to some given metric. Assume, furthermore, that the agent selects tasks to practice uniformly at random and without replacement, and that it practices a selected task until its current policy π can

3. A more expressive representation is needed for the joint policy π_{aug}, compared to that of a single option policy. Intuitively, π_{aug} needs to be able to capture all non-trivial correlations between task and state features and how they jointly determine optimal actions. Conversely, policies for a single option can be represented with fewer parameters because they only model how state features (but not task features) determine actions;

4. Policies for different problems cannot be easily learned in parallel and later combined to accelerate the construction of a skill; instead, an agent learning a single joint policy needs to practice one task at a time;

5. If the distribution of tasks changes, it is not possible to easily readjust the (single) learned joint policy to deal with the new distribution;

6. Specifying the task to be executed via additional state features assumes that the agent can arbitrarily assign values to those features. Most learning algorithms,

no longer be improved. Let c_0 be the expected number of samples required to learn an optimal policy, from scratch and starting from a fixed initial policy π_0, for one arbitrary decision-making problem. Assume that if the agent consecutively practices tasks that belong to the same cluster, it needs to collect only a small number p of samples; assume, conversely, that if the agent consecutively practices tasks that belong to different clusters, it undergoes negative transfer and requires a larger number P of samples in order to update its current policy to the subsequent task. Assume that $p \ll c_0 < P$; that is, consecutively solving problems that belong to a same cluster requires fewer training samples than solving a problem from scratch (i.e., by using as a starting point a fixed, initial policy π_0); and that learning from scratch, in turn, requires fewer samples than solving the problem but starting with an inappropriate policy (e.g., one for solving a problem from another cluster). Let $C(t, g)$ be the expected cost, in terms of number of samples, for an agent to solve t remaining decision-making problems given that it has selected a task from a cluster with g yet-unsolved problems. C can be expressed via the following set of recurrence relations: $C(t, t) = tp$; $C(t, 0) = P + C(t-1, t-1)$; and $C(t, g) = \frac{g}{t}[p + C(t-1, g-1)] + \frac{t-g}{t}[P + C(t-1, t-g-1)]$. Consider, as an example, the case when $n = 4$ and $G = 2$; it is possible to show that $C(4, 2) = c_0 + \frac{5}{2}P + \frac{3}{2}p$, where c_0 is the cost incurred in solving the first sampled task using as starting point the initial policy π_0. Consider, by contrast, the expected cost incurred by an agent that learns *independent* policies for each task, and that always uses as starting point the fixed initial policy π_0. In this latter case, the agent incurs an expected cost of $4c_0$ samples. Since $p \ll c_0$, the cost of learning independent policies is lower than that of learning a single joint policy if $4c_0 > c_0 + \frac{5}{2}P \Rightarrow P > 1.2c_0$. In other words: if the cost P resulting from negative transfer is 20% higher than the cost of solving a problem from scratch, it is possible to learn n independent policies faster than a single joint policy defined over n tasks.

however, assume that agents can *observe* their current states, but not that they can *actively set* new values to particular state features;

7. Specifying the task to be executed by π_{aug} via additional state features limits the learning algorithms that can be used. Black-box policy improvement methods [71], for instance, often ignore state information. Because the rewards received by an agent are task-dependent, and tasks are encoded via features that the algorithm may ignore, learning a joint policy over n tasks might become a non-stationary problem.

2.2.3 Grupen's Motor Control Framework

Grupen's motor control framework aims at constructing hierarchical closed-loop motor control plans [13]. In this framework actions are defined by three sets: motor resources Ω_τ, feedback signals Ω_σ and potential functions Ω_ϕ. Motor resources and feedback signals are grounded in the robot's sensor and actuator sets, respectively, and potential functions describe the objectives of the controller. The motor commands used to achieve a controller's goal are computed via the Jacobian $\frac{\partial\phi(\sigma)}{\partial\tau}$ of the potential function with respect to the motor output parameters. Controllers $c(\phi, \sigma, \tau)$ are defined by combining potential functions, feedback signals and motor variables and can be used as primitive actions by the robot. Planning occurs on a higher-level discretized state space in which states are predicates describing the current condition of each controller—for example, whether it is running, has converged or is in an unknown state. This framework effectively defines a control basis that can be used to construct hierarchical motor control plans and is strongly related to the options framework, though it was proposed first [28].

Because controllers can be bound to different sensorimotor couplings (i.e., different sets of sensor inputs and motor outputs) the control basis framework is particularly well-suited to transfer and generalization in robot systems. Hart et al. [25] achieve generalization in two steps. First, a policy is converted into an abstract form called a *declarative policy*. Declarative policies are alternative representations of a controller that allow it to be bound to different sources of sensor inputs and use different output effectors. A declarative policy captures only abstract information about the objectives required to meet a given behavioral goal. Assume, for instance, a controller for reaching objects. If this controller requires the cartesian coordinates of the target object, its declarative form will correspond to an abstract representation that explicitly states that the controller can use any input sources that provide said cartesian coordinates—for instance, laser scanners or a stereo vision system.

The second step in the generalization is achieved by computing and applying a *procedural policy*. A procedural policy is a function constructed via supervised learning that maps contexts (sets of features describing a problem or situation) to resources. Procedural policies parameterize a declarative policy with different sets of resources and thus allow for abstract controllers that can be used in different situations requiring different types of input data or output effectors. A procedural policy for the reaching task, for example, might encode that in some contexts it is advantageous for the agent to reach using its left arm instead of its right arm.

The type of abstraction and generalization made possible by the control basis is related to the generalization we aim for with parameterized skills. In our setting, however, we assume that the resources available to the robot (e.g., its sensors and actuators) are fixed—the robot always has full access to the complete set of resources. Furthermore, the flexible skills that we wish to construct parameterize not the robot's input and output resources, but the internal representation of the controller itself. This is equivalent to a potential function that can change according to the context. In this sense our objectives are complementary to those of the control basis and could be used to define an extra layer of parameterization—specifically, one by which the context modifies the internal policy used by lower-level controllers.

2.2.4 Methods Based on Transfer Learning

Parameterized skill learning may be seen as the problem of transferring policies designed for one task to different but possibly related tasks. Transfer Learning (TL) is a machine learning paradigm that focuses on tackling this problem. A few TL methods have been developed to address questions similar to that of learning parameterized skills.

Taylor and Stone [76] proposed a method for summarizing a source task policy as a set of abstract production rules, and leveraging those rules to more rapidly learn a target task. Their approach constructs a decision tree from trajectories (sampled from a known policy) and subsequently uses a translation function to transfer and apply the learned rules on a new task. A mapping between states and actions from the source task to a target task is assumed to be known. Liu and Stone [44] described a similar technique for transferring value functions between given pairs of tasks, but require prior knowledge of the tasks' dynamics in the form of a Dynamic Bayesian Network. Both of these methods are applicable if exact relations and mappings between source and target problems are known. They are not capable, however, of learning general parameterized skills that can tackle families of related motor tasks.

Value functions can also be transferred between different but related tasks. Taylor et al. [75] transfer Q-values between tasks defined over different state and action spaces. Their method constructs a translation function between the state and action representations of a source and target tasks by exhaustively searching over all possible mapping functions. The selected mapping corresponds to the best match between the transition models of a given pair of source and target tasks. Barrett et al. [3] introduced a similar method, in which the weights used to represent a Q-function are transferred to a target task. A mapping between state features of both tasks is required. It is also possible to directly transfer policies between tasks. Instead of transferring Q-functions, it is also possible to transfer policies directly. If a policy is represented, for example, as a neural network, its weights and structure may be reused to execute similar tasks by assuming that a mapping between features of the source and target tasks is available [77].

2.2.5 Methods Based on Directly Adjusting Existing Policies

Unlike methods based on Transfer Learning, other techniques exist that directly learn flexible policies. Schaal et al. [59] introduced Dynamic Movement Primitives, or DMPs, which are a framework for modular motor control based on a set of linearly-parameterized autonomous non-linear differential equations. The time evolution of these equations defines smooth kinematic control policies that can be used to drive a system. The trajectories encoded by a DMP can be obtained by integrating the following set of differential equations:

$$\kappa \dot{v} = K(g - x) - Dv + (g - x_0)f$$

$$\kappa \dot{x} = v,$$

where x and v are the position and velocity of the system, respectively; x_0 and g denote the start and goal positions; κ is a temporal scaling factor; and K and D act like a spring constant and a damping term, respectively. Finally, f is a non-linear function that can be learned in order to allow the system to generate arbitrarily complex movements and is defined as

$$f(s) = \frac{\sum_i w_i \psi_i(s) s}{\sum_i \psi_i(s)} \tag{2.4}$$

where $\psi_i(s) = \exp(-h_i(s - c_i)^2)$ are Gaussian basis functions with adjustable weights w_i and that depend on a phase variable s. The phase variable is constructed so that it monotonically decreases from 1 to 0 during the execution of a movement and is typically computed by integrating $\kappa \dot{s} = -\alpha s$, where α is a pre-determined constant. PID controllers are typically used to track such trajectories, especially in robotics applications. The weights **w** in the above-mentioned equation can be estimated either via direct policy search or Learning from Demonstration (LfD) techniques [53].

DMPs are capable of encoding flexible policies because after a set of weights **w** is learned, both the initial state x_0 and the goal state g of a desired trajectory may be specified. This allows DMPs to generate novel trajectories, which are qualitatively similar to the one observed when training but with different endpoints.

Although DMPs constitute a type of parameterized skill, they are not general enough to represent arbitrary reusable skills. This results, in part, from the fact that the number and nature of their tunable parameters (or *metaparameters*) are fixed and pre-determined. DMPs cannot be used to construct behaviors that depend on arbitrary task parameterizations; some families of related tasks, for instance, might require parameterizations that depend on more than just initial and goal states in a trajectory. Furthermore, different types of tasks may require the use of qualitatively different trajectories. As an example, consider a tennis player that uses parameterized forehand and backhand movements to hit the ball. DMPs are not capable of jointly modeling the different trajectories needed to execute different ball-hitting movements; that would require a set of task-dependent weight vectors **w**. Finally, note that even for those tasks that can be parameterized solely by initial and goal states, it is not always clear how to set those metaparameters given a high-level description of the task to be performed.

In order to address some of these limitations, Kober et al. [33] proposed learning a mapping from task description to metaparameters of a DMP. This mapping is constructed via cost-regularized kernel regression, from which it is possible to estimate both the mean predicted metaparameter appropriate for a task and the uncertainty with which it executes that task. Their method was developed specifically for DMPs and does not support other types of policies; furthermore, it assumes that DMP metaparameters are sufficient to represent all possible classes of parameterized tasks of interest. Bitzer et al. [10] take a different approach and synthesize novel robot movements by modulating DMPs learned in a latent space and projecting them back onto the original pose space of robot. Their method does not support arbitrary task parameterizations and is not capable of addressing task families that require the use of more than one qualitatively different trajectory.

Other methods exist for directly adjusting existing policies. Soni and Singh [65] introduced a technique for creating adjustable options whose termination criteria is adapted, on-the-fly, to deal with unknown, changing aspects of a motor task. This technique does not, however, directly predict a complete parameterization of appropriate policies for novel tasks. Hoffmann et al. [27] introduced a type of parameterized model designed to deal with continuously changing environments. They assume that parts of the environment (e.g., the mass of an object with which the robot interacts) cannot be observed and are under constant change, and thus need to be estimated or indirectly taken into account by the skill. Their method also does not construct skills which can be directly parameterized by an agent, and does not work under arbitrary policy representations.

Finally, Hausknecht and Stone [26] presented a method for learning a parameterized skill for kicking a soccer ball with varying amounts of force. They exhaustively vary one particular policy parameter known *a priori* to be relevant for the skill and measure the resulting distance traveled by the ball. By assuming a quadratic relation between these variables, they are able to construct a regression model and invert it, thereby obtaining a closed-form expression for the policy parameter value required for a desired kick. This is an interesting example of the type of parameterized skill that we would like to construct, albeit a very domain-dependent one.

2.2.6 Discussion

Previous research has considered the problem of learning flexible skills largely in isolation and has not addressed the construction a general cohesive framework for parameterized skill acquisition. Few existing approaches are agnostic regarding the type of policy parameterization used, which further limits their applicability and constrains the types of skills that can be learned. Recent work on DMPs focuses primarily on open-loop policies and typically assumes that their metaparameters are sufficient to represent arbitrary families of motor tasks. Finally, most of the existing transfer learning approaches assume that mappings between features of a source and target tasks are known *a priori*. In this dissertation we introduce a general framework for learning parameterized skills that addresses these limitations and that complements existing methods by adding capabilities for identifying and modeling useful sub-skills, and for actively designing training regimens that result in more rapid skill acquisition.

CHAPTER 3
LEARNING PARAMETERIZED SKILLS

In this chapter we introduce a general framework for learning parameterized skills. Our main objective is to design a method by which reusable skills can be learned from limited data and subsequently used to solve a distribution of related problems. Our method solves a small number of sample decision-making problems and uses the corresponding learned policies to estimate properties of the low-dimensional manifold on which they lie. This manifold models how policy parameters change as we vary the parameters of a problem. We combine manifold learning techniques, classification algorithms and non-linear regression methods to estimate geometric and topological properties of such a manifold, thereby obtaining a general model by which policies can be predicted given only a description of the problem to be solved.

3.1 Setting and Objective

Consider an agent presented with a sequence of decision-taking problems (which we henceforth interchangeably refer to as *tasks*) drawn from some distribution of problems. Assume that each problem is modeled as an MDP, and that MDPs have dynamics and reward functions similar enough so that they can be considered variations of a same overall problem. Consider, as an example, a robot tasked with moving different objects in a warehouse. While doing so it may have to pick up objects of different shapes and weights. Each time that it is presented with a new object, the robot needs to determine an appropriate grasping behavior. This can be achieved by computing the optimal policy for an MDP specifically constructed to reflect the properties of that particular object. States in this MDP could encode, for instance, the current configuration of the robot's body and visual features of the scene and of object. Large rewards may be given for achieving stable and successful grasping configurations. The robot, then, while interacting with different objects in the warehouse, needs to solve a *sequence* of MDPs. Since these MDPs encode variations of a same overall problem—namely, grasping—we assume that they have similar transition dynamics and reward functions.

In order to deal with sequences of related problems of this kind, agents might choose to solve each one independently and from scratch. Alternatively, they may exploit the fact that the problems have similar structure and learn a single parameterized skill, which they subsequently use to produce solutions to novel problems given only a description of the problem.

Let Ψ be the set of possible decision-making problems that an agent might need to solve. Each element of this space is an MDP, which we assume can be compactly described by a vector τ of parameters. Learning to grasp a particular object, for instance, is a decision-making problem that may be compactly described by parameters specifying that object's shape and weight. Assume, furthermore, that problems in Ψ occur in an agent's lifetime with probabilities given by some distribution P. The objective of a parameterized skill is to maximize the agent's expected reward over the entire distribution of possible problems:

$$\int P(\tau) J\big(\pi_\theta, \tau\big) d\tau, \tag{3.1}$$

where $\tau \in T$ is a vector of parameters compactly describing a particular problem in Ψ, T is the space of possible compact problem descriptions, π_θ is a policy with parameters $\theta \in \mathbb{R}^M$, and $J(\pi_\theta, \tau) = E\big[\sum_t r_t | \pi_\theta, \tau\big]$ is the expected return obtained when using policy π_θ to solve problem τ. Finally, $P(\tau)$ is the probability with which problem τ occurs during the agent's lifetime; concretely, let $\mathcal{T} : \Psi \rightarrow T$ be a random variable mapping each possible decision-making problem in Ψ to its corresponding compact description τ. Then, $P(\mathcal{T} = \tau)$ is the probability that the agent has to solve a particular problem with parameters τ. Because we use *problem* and *task* interchangeably, we often refer to T as a *task space*, that is, the space of possible *task parameters* that compactly describe elements of Ψ.

We now define a *parameterized skill* as a function

$$\Theta : T \to \mathbb{R}^M \tag{3.2}$$

mapping task parameters to policy parameters. Concretely, a skill maps the compact description $\tau \in T$ of a problem in Ψ to an appropriate policy with parameters $\Theta(\tau) \in \mathbb{R}^M$. The optimization objective of a parameterized skill is to maximize Equation 3.1, and may be written as:

$$\max_{\Theta} \int P(\tau) J\big(\pi_{\mathbf{\Theta}(\tau)}, \tau\big) d\tau. \tag{3.3}$$

A parameterized skill can be constructed via a training set of sample decision-making problems and their corresponding optimal or near-optimal policies. In this thesis we assume that a skill-learning agent has access to (or can construct) a set K of pairs $\{\tau, \theta_\tau\}$, where $\tau \in T$ is the parameter vector describing a particular problem in Ψ and θ_τ are the parameters of an optimal or near-optimal policy for solving it. By analyzing K, an agent may attempt to model how policy parameters change as the parameters τ of a problem are varied. This model, once learned, can be used to predict appropriate policies for novel problems given only their corresponding compact descriptions τ.

3.2 Assumptions

We make four central assumptions in this thesis. First, we assume that similar decision-making problems generally have similar optimal or near-optimal policies. In particular, policies for solving MDPs in Ψ might have similar parameters since the problems that they solve have similar transition and rewards dynamics.

Secondly, we assume that the set of policies for solving problems in Ψ has a structure that can be exploited in order to learning a parameterized skill. Consider policies π_θ with parameters $\theta \in \mathbb{R}^M$. These policies may be seen as points in an M-dimensional space, which we henceforth refer to as *policy space*. Even though they have M parameters which can, in principle, assume arbitrary values, we argue that policies for problems in Ψ do not truly have M degrees-of-freedom. To see why, note that the problems solved by these policies can vary in only $\dim(T)$[1] ways—concretely, by varying each one of the $\dim(T)$ parameters of $\tau \in T$. Since policies may have fewer than M degrees-of-freedom, and because generally $\dim(T) \ll M$, we expect them to be confined to a lower-dimensional manifold[2] embedded in $\mathbb{R}^M$. The fact that the policy space has this kind of structure can be exploited by parameterized skill learning algorithms to more easily predict appropriate policies for novel problems.

Thirdly, we assume that smooth changes to the parameters τ of a problem result in smooth changes to its policy. If that is the case, it is possible to smoothly move over the manifold of policies by smoothly varying task parameters. This assumption is reasonable in a variety of situations, especially in the common case where a policy π_θ is differentiable with respect to its parameters. A motivating example to this assumption is discussed in Appendix A.

Fourthly, we assume that even though similar problems generally have similar solutions (see Assumption 1), problems may in exist Ψ that, if slightly modified, result in qualitatively different policies. Consider, for instance, how particular changes to the speed of a horse may cause it to adopt qualitatively different gaits. A walking

[1]Here, $\dim(\mathcal{S})$ denotes the dimensionality of space $\mathcal{S}$.

[2]Manifolds are topological spaces that locally resemble Euclidean space; each point on a p-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension p.

gait is used when moving at speeds less than 4 miles per hour. In this gait, the horse always keeps one foot raised and the other three on the ground. At approximately 5 miles per hour, a qualitatively different gait is adopted—trotting—during which the horse moves its legs in unison in diagonal pairs. Finally, as a horse approaches approximately 12 miles per hour it adopts a third gait: galloping. The need for qualitatively different behaviors as a function of the speed suggests that particular changes to the parameters of a problem—such as increasing speed from 5 to 6 miles per hour—may result in abrupt changes to its policy.

Motivated by example above, we divide our fourth assumption into two more specific assumptions. First, we assume that problems with *parameters within certain subregions* of the task space T have similar policies. Because these policies have similar parameterizations, they lie on a single smooth surface embedded in $\mathbb{R}^M$. Secondly, we assume that problems with parameters in *other* regions of the task space may require significantly different policies, which results in those policies being embedded in a different lower-dimensional chart. In general, we expect the policies for problems in Ψ to be distributed over a set of disjoint, piecewise-smooth lower-dimensional surfaces, or *charts*. Separate charts exist whenever particular changes to the parameters of a problem result in a discontinuous change to the parameterization of its policy. Each individual chart typically models policies for solving a subclass of related problems, such as *galloping* at speeds between 12 and 30 miles per hour might. A different subclass of related problems, such as *walking* at speeds between 0 and 4 miles per hour, might have their policies embedded in a separate lower-dimensional chart.

3.3 A Parameterized Skill Model

In this section we introduce a general algorithm for learning parameterized skills. Our method begins by constructing a set K of sample decision-making problems and corresponding learned optimal or near-optimal policies. We assume that the sample problems τ used in the construction of K are drawn from the task distribution P. Formally, let $K \equiv \{(\tau_i, \theta_i)\}$, where $\tau_i \in T$ is the parameter vector describing the i-th training task drawn from P and $\theta_i = (\theta_i(1), \ldots, \theta_i(M))^\top \in \mathbb{R}^M$ are the M parameters of a policy solving it. Our algorithm uses the set K of sample tasks and corresponding solutions to construct a family of regression models (one per policy parameter) mapping task parameters to policy parameters. These models allow an agent to identify where (in the policy space) the solution to a particular problem is. In this sense, they can be seen as a unified model of the smooth manifold (embedded in $\mathbb{R}^M$) where task policies lie. Since policies for different tasks might be embedded in different low-dimensional charts, our algorithm first estimates the number D of charts within the manifold and only then constructs a specialized model of each one.

Our method consists of four general steps:

1. construct a skill training set K by drawing sample problems from P and learning an optimal or near-optimal policy for solving them;

2. use K to estimate properties of the geometry and topology of the policy manifold. Specifically, our algorithm *a)* estimates the number D of lower-dimensional charts that compose the policy manifold; and *b)* constructs a training set K_χ (to be used in step 3) of pairs (τ, c), where τ is a training task in K and $c \in \{1, \ldots, D\}$ is an integer number identifying the chart in which that task's policy is embedded;

3. use K_χ to train a classifier $\chi : T \to \{1, \ldots, D\}$ mapping task parameters $\tau \in T$ to $\{1, \ldots, D\}$. χ allows for the identification of the chart where a given task's policy is most likely embedded;

4. train a set of $M \times D$ independent non-linear regression models $\Phi_{i,j}$, $i \in \{1, \ldots, D\}$, $j \in \{1, \ldots M\}$, each one mapping elements of T to individual policy parameters

θ_i, $i \in \{1, \ldots M\}$. Each subset $\{\Phi_{i,1}, \ldots, \Phi_{i,M}\}$ of regression models is trained over all tasks τ in K where $\chi(\tau) = i$.[3]

This training process above results in D sets of M non-linear regressions models. Each set contains the models necessary for predicting policy parameters, given task parameters, for problems whose policies are embedded in a same given chart. These models are then combined to construct a parameterized skill function $\Theta(\tau)$:

$$\Theta(\tau) \equiv (\Phi_{\chi(\tau),1}, \ldots, \Phi_{\chi(\tau),M})^T. \tag{3.4}$$

Using a learned parameterized skill to solve novel problems involves three main steps, depicted in Figure 3.1: *1)* task parameters τ are drawn from the task distribution P; *2)* the classifier χ is used to identify in which chart that task's policy is embedded; *3)* the corresponding set of regression models associated with that chart is used to predict policy parameters for the task.

Note that the algorithm introduced above makes no assumptions regarding which policy representation, manifold analysis method, and classification and non-linear regression techniques are to be used. These are design decisions best made in light of the characteristics of the application at hand. In the following section we introduce one possible instantiation of this framework, in which an underactuated robotic arm is tasked with learning to accurately throw darts at parameterized target locations.

[3]This last step assumes that the policy features are approximately independent conditioned on the task. If this is known not to be the case, it is possible to alternatively train a set of D multivariate non-linear regression models Φ_i, $i \in \{1, \ldots, D\}$, each one mapping elements of T to complete policies parameterizations $\theta \in \mathbb{R}^M$, and use them to construct Θ. Again, the i-th such model should be trained only over tasks τ in K such that $\chi(\tau) = i$. The problem of predicting a vector of possibly correlated policy features can also be addressed via structured prediction techniques [84].

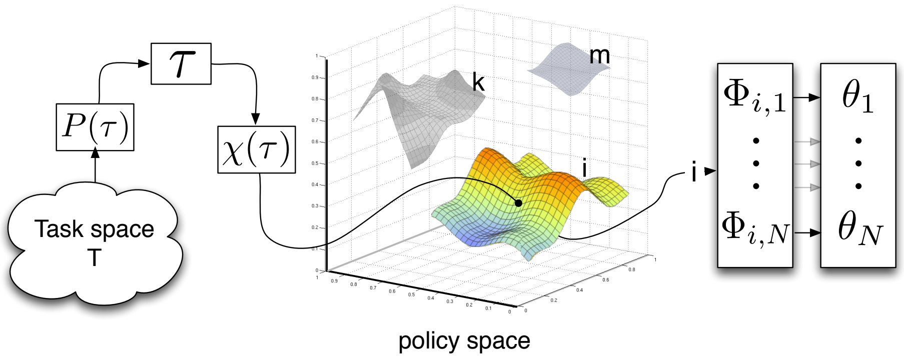

Figure 3.1. Steps involved in executing a parameterized skill. First, a task is drawn from the distribution P; the classifier χ identifies the chart to which the policy for that task belongs; finally, the corresponding regression models map task parameters to policy parameters.

3.4 The Parameterized Dart Throwing Domain

In the parameterized dart throwing problem, a simulated planar underactuated robotic arm is tasked with learning a skill to accurately throw darts at parameterized target locations around it (Figure 3.2). The base of the arm is affixed to a wall in the center of a 3-meter high and 4-meter wide room. The arm is composed of three connected links and a single motor which applies torque only to the second joint, making this a difficult non-linear and underactuated control problem. At the end of its third link, the arm has an actuator capable of holding and releasing a dart[4]. The state of the system is a 7-dimensional vector composed of 6 continuous features corresponding to the angle and angular velocities of each link and by a seventh binary feature specifying whether or not the dart is still being held. The goal of the system is to control the arm so that it executes a throwing movement and accurately hits a target of interest. In this domain the space T of tasks consists of a 2-dimensional

[4]We designed this problem so that the three links have combined mass almost negligible compared to the mass of the dart; their main purpose it to make the system harder to control. The link directly affixed to the wall is a 50cm $\times$ 2.5cm rectangle with mass of 15 grams; the link connected to it is a 35cm $\times$ 2.5cm rectangle with mass of 12 grams; and the final link is a 25cm $\times$ 2.5cm rectangle with mass of 8 grams. The dart is modeled as a circle with 10cm of radius and mass of 200 grams. The maximum torque allowed is 2.75 Nm.

Euclidean space containing all (x, y) coordinates at which a target can be placed—a target can be affixed anywhere on the walls or ceiling surrounding the agent.

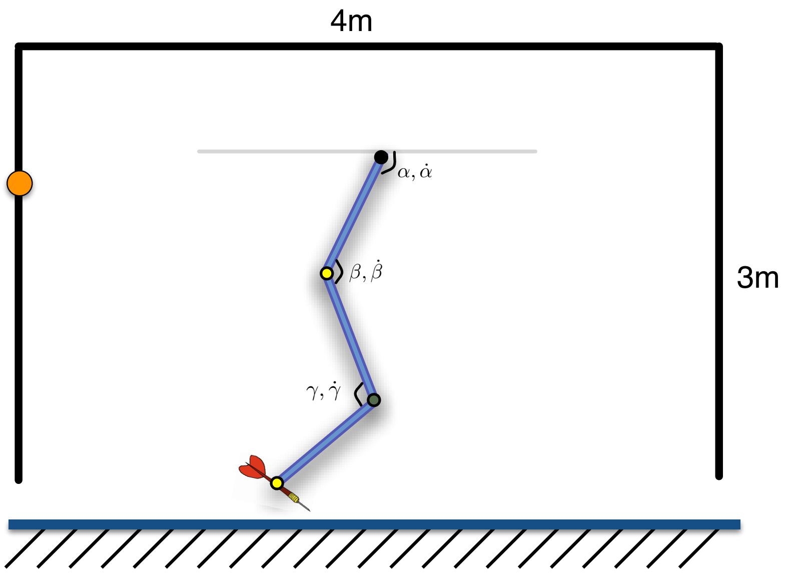

Figure 3.2. The dart throwing domain (not to scale). One possible target is depicted as a circle on the wall to the left of the robot.

To obtain a concrete instantiation of the algorithm outlined in Section 3.3 we need to specify *1)* a policy representation; *2)* a policy-learning algorithm; *3)* a method to analyze the policy space and estimate D, the number of lower-dimensional charts on which task policies lie; *4)* a method to construct the non-linear classifier χ; and *5)* a method to construct the set of non-linear regression models Φ. In this section we introduce and discuss the specific algorithms and techniques chosen to tackle the parameterized dart-throwing domain. We discuss experiments and results in Section 3.5.

Our choices of methods are directly guided by the characteristics of this domain. Because it involves a multi-joint simulated robotic arm, we use a policy representation that is well-suited to robotics: Dynamic Movement Primitives, or DMPs (see Section 2.2.5). We use a DMP to model the trajectory of torques to be applied to controlled joint of the arm. A PID controller is used to tracking these trajectories. The DMP-based policy we obtain is parameterized by a 37-dimensional policy feature vector $\theta = (\lambda, g, w_1, \ldots, w_{35})^T$, where λ is the value of the DMP phase variable s at which the arm should let go of the dart; g is the goal parameter of the DMP; and $w_1, \ldots, w_{35}$ are the weights of each Gaussian basis function in the movement primitive (Equation 2.4).

We combine DMPs with a policy learning algorithm known to perform well with this type of policy representation. PoWER [31] is a policy search method that collects sample path executions from a policy and updates policy parameters towards ones that induce a new success-weighted path distribution. We use PoWER due to its simplicity and because it has been empirically shown to outperform other policy learning algorithms in a variety of standard benchmarks and on real robotics problems [32]. PoWER works by executing rollouts ρ constructed based on slightly perturbed versions of the current policy parameters; these perturbations consist in a structured, state-dependent exploration ${\varepsilon_{\mathbf{t}}}^T \phi(\mathbf{s}, t)$, where $\varepsilon_{\mathbf{t}} \sim \mathcal{N}(0, \hat{\mathbf{\Sigma}})$ and $\hat{\mathbf{\Sigma}}$ is a meta-parameter of the exploration; $\phi(\mathbf{s}, t)$ is the vector of policy feature activations at time t. By adding this type of perturbation to θ we induce a stochastic policy whose actions are $a = (\theta + \varepsilon_{\mathbf{t}})^T \phi(\mathbf{s}, t)) \sim \mathcal{N}(0, \phi(\mathbf{s}, t)^T \hat{\mathbf{\Sigma}} \phi(\mathbf{s}, t))$. After performing rollouts using such a stochastic policy, the policy parameters are updated as follows:

$$\begin{aligned}\theta_{k+1} \quad &= \quad \theta_k + \left\langle \sum_{t=1}^{T} \mathbf{W}(\mathbf{s}, t) Q^{\pi}(\mathbf{s}, \mathbf{a}, t)) \right\rangle_{\omega(\rho)}^{-1} \times \\ &\quad \left\langle \sum_{t=1}^{T} \mathbf{W}(\mathbf{s}, t) \varepsilon_t Q^{\pi}(\mathbf{s}, \mathbf{a}, t)) \right\rangle_{\omega(\rho)}\end{aligned}$$

where $\hat{Q}^{\pi}(\mathbf{s}, \mathbf{a}, t) = \sum_{\tilde{t}=t}^{T} r(\mathbf{s}_{\tilde{t}}, \mathbf{a}_{\tilde{t}}, \mathbf{s}_{\tilde{t}+1}, \tilde{t})$ is an unbiased estimate of the return, $\mathbf{W}(\mathbf{s}, t) = \phi(\mathbf{s}, t) \phi(\mathbf{s}, t)^T \left(\phi(\mathbf{s}, t)^T \hat{\mathbf{\Sigma}} \phi(\mathbf{s}, t) \right)^{-1}$ and $\langle \cdot \rangle_{\omega}(\rho)$ denotes an importance sampler which can be selected based on the domain. A useful heuristic when defining ω is to discard sample rollouts with very small importance weights. In the experiments discussed in

Section 3.5 we use importance weights proportional to the relative performance of a rollout in comparison to others.

To analyze the geometry and topology of the policy space and estimate the number D of lower-dimensional charts we use the ISOMAP algorithm [78]. ISOMAP is a technique for learning the underlying global geometry of high-dimensional spaces and the number of non-linear degrees of freedom that underlie it. It allows an agent to estimate the number of disjoint charts in policy space and to identify to which chart a given training policy belongs. We use this information to train a classifier χ mapping task parameters $\tau \in T$ to $\{1, \ldots, D\}$. This classifier is subsequently used to identify the lower-dimensional chart where a given task's policy is most likely embedded. In this instantiation of the parameterized skill framework, we implement χ as a simple linear classifier.

Finally, we must choose a non-linear regression method to construct $\Phi_{i,j}$. We use Support Vector Regression (SVR) [20] due to their good generalization capabilities and relatively low dependence on parameter tuning. In the experiments discussed in Section 3.5 we use SVRs with Gaussian kernels and a inverse variance width of 5.0. As previously mentioned, if important correlations between policy and task parameters are known to exist, multivariate regression models (such as Structured Support Vector Machines [81]) might be preferable.

3.5 Experiments

Before discussing the performance of parameterized skill learning in the dart-throwing domain, we analyze and discuss important topological characteristics of the space of policies induced as we vary task parameters. We sampled 60 tasks (target positions) uniformly at random and placed target boards at the corresponding locations. Near-optimal policies for executing each one of these tasks were learned using PoWER; the learning algorithm was configured to perform a policy update every 20 rollouts and to run until a *minimum performance threshold* was reached. In our experiments, this criterion corresponded to the moment when the robotic arm first executed a policy that landed the dart within 5 centimeters of the intended target. In order to speed up the sampling process we initialize policies for subsequent targets with ones computed for previously sampled tasks.

We first analyze the structure of the induced policy manifold; this is done by modeling how each dimension of a policy varies as we smoothly vary the task. Figure 3.3a presents this information for a representative subset of policy parameters. On each subgraph of Figure 3.3a the x axis corresponds to a 1-dimensional representation of the task obtained by computing the angle at which the target is located with respect to the arm. This is done for ease of visualization, since using x, y coordinates would require a 3-D figure. The y axis corresponds to the value of a selected policy parameter. The first important observation to be made is that as we vary the task, not only do policy parameters vary smoothly, but they tend to remain confined to one of two disjoint but smoothly varying lower-dimensional surfaces. A discontinuity exists, indicating that after a certain point in task space a qualitatively different policy parameterization is required. Another interesting observation is that this discontinuity occurs approximately at the task parameter values corresponding to hitting targets directly above the robotic arm. This implies that policies for hitting targets to the left of the arm lie on a different manifold than policies for hitting targets to its right[5]. This is relevant for two reasons: *1)* it confirms that the manifold assumption is reasonable and that smooth task variations induce smooth, albeit non-linear, policy changes; and *2)* it shows that the policies for solving a distribution of problems are generally confined to one of several lower-dimensional charts, and that the way

[5]In fact, the existence of disjoint charts in this case results from a type of symmetry in the policy space: the two identified charts encode throwing movements that are mirror reflections of each other, and that are represented using qualitatively different policy parameterizations.

in which they are distributed among these charts is correlated to the qualitatively different strategies that they implement.

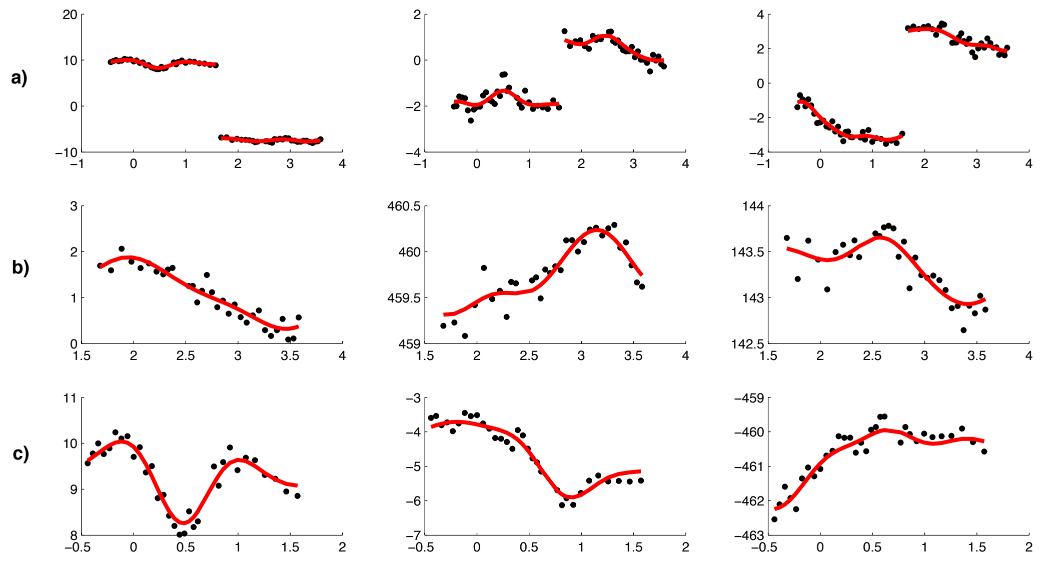

Figure 3.3. Analysis of the variation of a subset of policy parameters as a function of smooth changes in the task.

Figures 3.3b and 3.3c show, similarly, how a selected subset of policy parameters changes as we vary the task, but now with the two resulting charts analyzed separately. Figure 3.3b shows the variations in policy parameters induced by smoothly modifying tasks for hitting targets anywhere in the interval of 1.57 to 3.5 radians—that is, targets placed roughly at angles between 90° (directly above the agent) and 200° (lowest part of the right wall). Figure 3.3c shows that same information but for targets located in one of the other two quadrants—that is, targets to the left of the arm. We superimposed in Figures 3.3a-c a red curve representing the non-linear fit Φ modeling the relation between task and policy parameters in each chart. Note how a clear linear separation exists between which task policies lie on which lower-dimensional chart: this separation indicates that two qualitatively distinct types of movement are required for solving different subsets of the tasks. Because we empirically observe that a linear separation exists, we construct χ as a linear classifier mapping tasks parameters to the numerical identifier of the chart in which that task's policy is embedded.

We can also analyze the characteristics of the lower-dimensional, quasi-isometric embedding of policies produced by ISOMAP. Figure 3.4 shows the 2-dimensional embedding of a set of policies sampled from one of the charts. Embeddings for the other chart have similar properties. Analysis of the residual variance of ISOMAP allows us to conclude that the intrinsic dimensionality of the policy manifold is 2; this is expected since we are essentially parameterizing a high-dimensional policy space by task parameters, which are drawn from the 2-dimensional space T. This implies that even though policies are part of a 37-dimensional space, they remain confined to a 2-dimensional manifold because there are just two degrees-of-freedom with which we can vary the tasks. In Figure 3.4 lighter-colored points identify embeddings of policies for hitting targets at higher locations. Note how each point in this embedding space is surrounded by points with similar colors[6]. This implies that policies for similar tasks (e.g., targets at similar heights) remain geometrically close in policy space—or, equivalently, that nearby targets have similar near-optimal policies.

Figure 3.5 shows three sample movements that the arm can execute in order to hit particular targets. Each inset figure shows the configuration of the arm at selected moments, $t_0, \ldots, t_4$, during a throw. We overlay current and past configurations of each link—more recent states of the arm are shown in lighter colors. The dart is depicted as an orange circle. By analyzing the trajectory of the dart during each throw, it is possible to observe that it follows a non-linear path in the configuration space of the robot. This results from the fact that the arm is underactuated and

[6]Note that the axes in Figure 3.4, representing two embedding dimensions of a given set of policies, have somewhat irrelevant units. This occurs because ISOMAP computes embedding coordinates so that pairwise Euclidean distance between embeddings approximate geodesic distances in the manifold; as such, the resulting embedding coordinates do not necessarily have a natural interpretation in terms of the original dimensions.

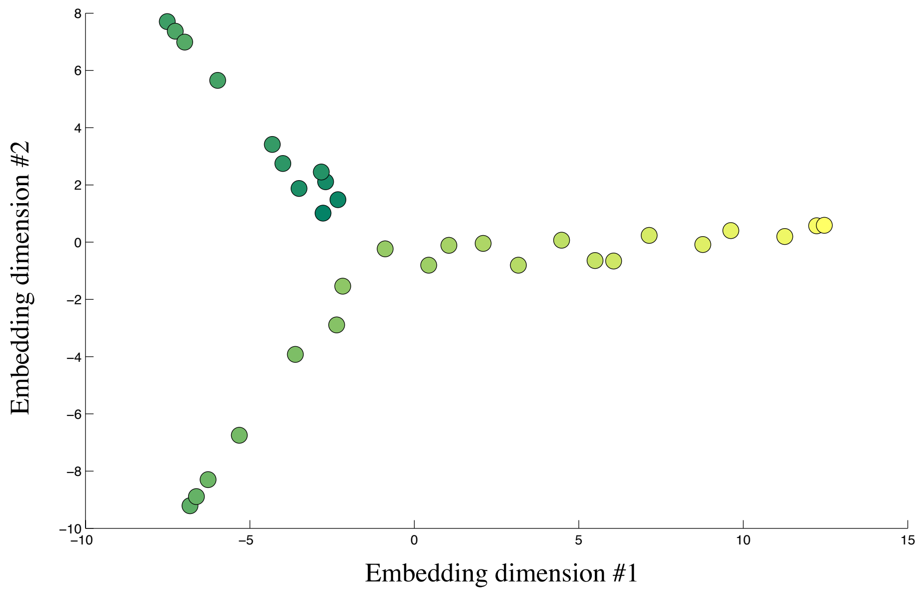

Figure 3.4. 2-dimensional embedding of policies parameters.

has to build momentum in order to hit certain targets. Figure 3.5a and Figure 3.5b show trajectories for hitting targets high on the ceiling and low on the right wall, respectively. These were used as training examples to construct a parameterized throwing skill. A total of five sample policies were used to train the skill; each predicted policy was further improved by allowing PoWER to execute two additional policy updates. Figure 3.5c shows a throwing movement produced by the skill for a novel task, corresponding to a target in the middle of the right wall.

Figure 3.6 shows the error in policy parameters predicted by the skill as a function of the number of examples used to train it. This is a measure of the relative error between the policy parameters predicted by Θ and parameters of a known solution for the same task. The lower the error, the closer the predicted policy is (in norm) to the correct solution. Figure 3.6 shows errors averaged over the parameters of 15 unknown tasks sampled uniformly at random from the task space. After 6 training samples are presented to the parameterized skill, it is already capable of predicting policies whose parameters are within 6% of the correct ones; with approximately 15

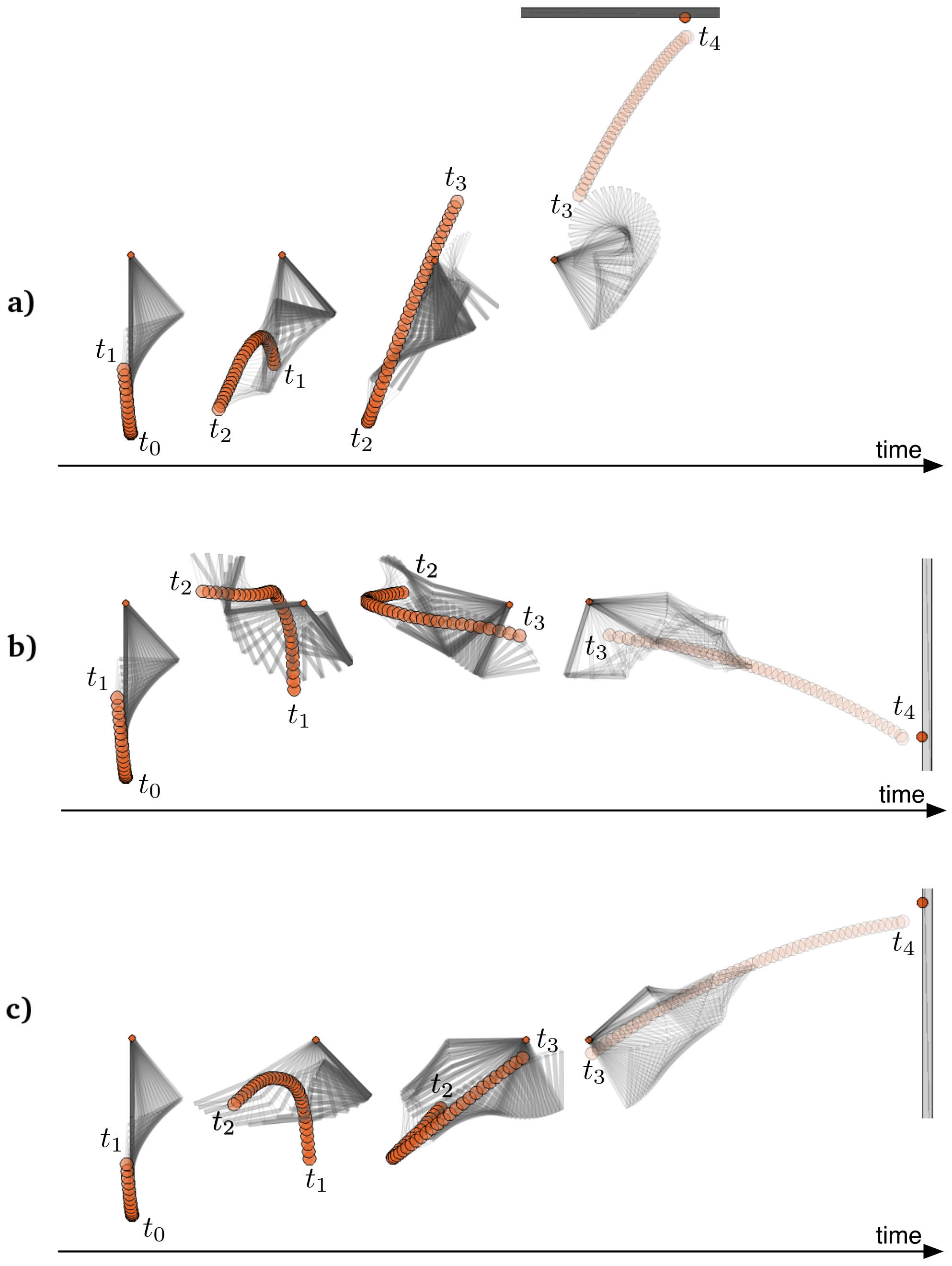

Figure 3.5. Learned parameterized arm movements. Arm movements (a) and (b) were used as training examples to the parameterized skill; (c) is the movement predicted by the skill for hitting a novel target.

samples, this error stabilizes around 3%. This accuracy is only possible because the policy space—even though high-dimensional—is also highly structured; specifically, because policies for similar problems lie on a lower-dimensional chart whose regular topology can be exploited when generalizing known solutions to novel tasks.

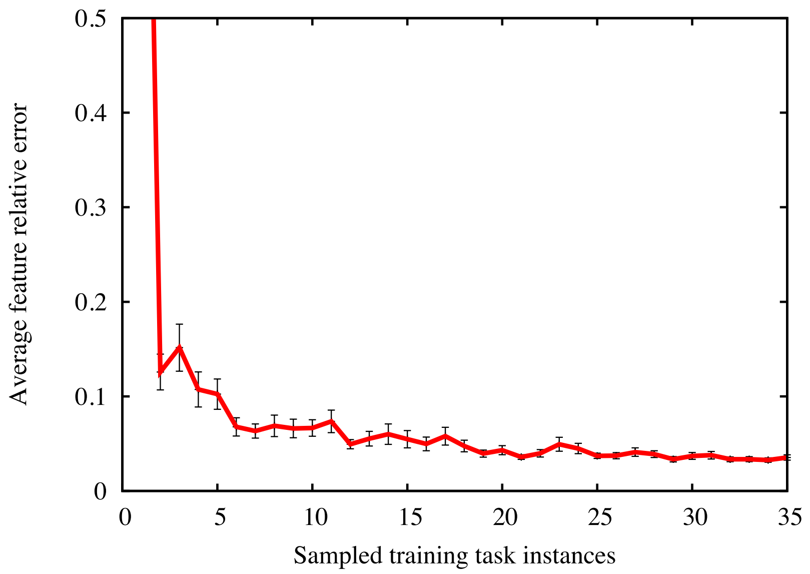

Figure 3.6. Average predicted policy parameter error as a function of the number of sampled training tasks.

Since some policy representations might be particularly sensitive to noise, we additionally measure the effectiveness of predicted policies when directly applied to novel tasks. Specifically, we measure the distance between the position where the dart hits and the intended target. This measurement is obtained by directly executing the predict policy and *before* any further learning or policy improvement takes place. Figure 3.7 shows that after 10 samples are presented to the skill, it can produce throws capable of hitting within 70cm of the intended target. This is a reasonable error if we consider that novel targets may be placed anywhere on a surface that extends for a total of *10 meters*. If the parameterized skill is presented with a total of 24 samples, the average error decreases to 30cm. This corresponds, roughly, to darts being thrown from 2 meters away and landing one dartboard away from the intended center.

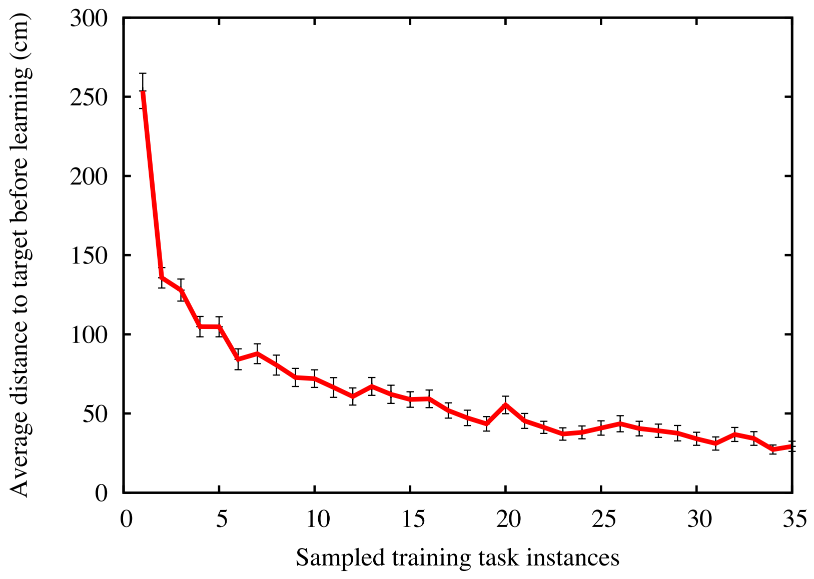

Figure 3.7. Average distance to target (*before any policy improvement*) as a function of the number of sampled training tasks.

Although these initial solutions are reasonable, especially considering that no direct learning with the target task took place, they are not perfect. We might therefore want to further improve them. Figure 3.8 shows how many additional policy updates are required to improve policies predicted by the skill up to a point where they reach our performance threshold—that is, landing the dart within 5 centimeters of the intended target. The dashed line in Figure 3.8 shows that on average 22.6 policy updates are required to find a near-optimal policy if the agent *does not use a skill* and has to practice each novel task from scratch. By using a parameterized skill trained with 9 examples, on the other hand, it is already possible to decrease this number to just 4 policy updates. If 20 training examples are used to construct the skill, it takes the agent only 2 additional policy updates to improve the predicted policy so that it meets the performance threshold.

These experiments demonstrate how useful a parameterized skill can be for agents facing sequences of related problems. Instead of repeatedly incurring the cost of learning to execute new tasks from scratch, it is more advantageous to use these experiences to construct a reusable skill capable of generalizing previous experiences to more rapidly tackle novel problems. A parameterized skill is capable, once learned, to directly produce appropriate policies for novel tasks without ever explicitly practicing those tasks. If the predicted solutions do not meet the agent's performance requirements, they can be further improved via standard policy search algorithms—usually at a fraction of the cost required to learn from scratch.

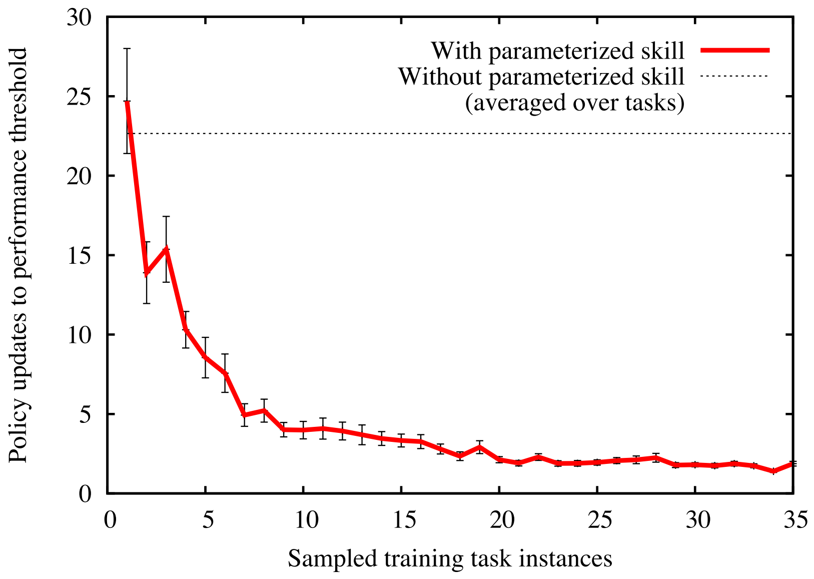

Figure 3.8. Average number of policy updates required to improve the solution predicted by the parameterized skill as a function of the number of sampled training tasks.

3.6 Related Work

Instead of explicitly learning a function to predict policy parameters for novel tasks, a naïve nearest-neighbor technique is also possible. In this case the agent maintains a library of known problem solutions and, when presented with a novel task, directly selects and executes the policy of the nearest known problem. This approach does not scale to complex families of parameterized problems because the number of observed task solutions required to uniformly cover the task space T grows exponentially with its dimensionality. Alternatively, techniques based on k-nearest neighbors regression are also possible. These correspond to non-parametric approaches that linearly interpolate the policies of the k nearest known problems. This type of approach may fail if task policies lie on a non-linear manifold. Furthermore, regression methods of this kind typically do not consider that different charts may exist. If that is the case, incompatible solutions (embedded in different charts) may be combined and produce policies that lie outside of the manifold of valid solutions. This is similar in nature to the problem of learning inverse kinematic solutions by averaging examples on a non-convex set [29]. A 2-joint robotic arm, for instance, may be able to reach a same object in two different ways, but if those solutions (when represented in joint space) are averaged, the arm misses the target object completely. This occurs because interpolated solutions in joint space do not necessarily yield a correct result in Cartesian space. Similarly, parameterized skill methods that linearly interpolate policies on a non-linear, multi-chart manifold may produce solutions that lie outside of the manifold of valid solutions.

Alternative, more sophisticated approaches have been proposed under the general heading of skill transfer. Konidaris et al. [36, 40] introduce a method for constructing reusable options by learning them in an agent-centered state space instead of in the original state space. This technique does not, however, construct generalized skills capable of solving a family of related tasks. Soni and Singh [65] create adaptable options whose meta-parameters, e.g., their termination criteria, can be adapted on-the-fly in order to deal with unknown, changing aspects of a task. However, this technique does not directly predict a complete parameterization of the policy for new tasks. Liu and Stone [44] propose a method for transferring a value function between a specific given pair of tasks but require prior knowledge of the task dynamics in the form of a Dynamic Bayesian Network. Finally, Braun et al. [12] discuss the idea of *structure learning* in humans, a concept that is very similar in nature to the one of parameterized skills. The authors present studies from cognitive and motor neuroscience that show that humans explicitly learn abstract structural invariants of their environments in order to facilitate generalization to novel tasks. They then introduce a Bayesian model with carefully designed priors capable of explaining experimental data supporting structure learning in humans. The authors do not, however, propose a concrete method for autonomously identifying invariants that allow for previous learning experiences to be generalized to novel tasks.

3.7 Discussion

We introduced a general framework for constructing parameterized skills. The idea underlying our method is to solve a small number of problem instances and generalize their solutions to new problems by combining manifold learning techniques, classification algorithms and non-linear regression methods in order to estimate geometric and topological properties of manifold. This allows the agent to obtain a general model by which policy parameters can be predicted from task parameters. This approach is effective because it exploits the intrinsic structure of the policy space—namely, that policies for similar tasks typically lie on lower-dimensional charts. Furthermore, our framework is capable of autonomously discovering the qualitatively different sub-skills required to execute distinct classes of tasks in a distribution.

This basic method can be extended in several important directions. First, the question of how to actively select training tasks to improve the overall readiness of a skill needs to be addressed. We introduce, in Chapter 5, an active skill learning technique. Another important open problem is that of how to properly deal with non-stationary distributions of tasks. If a new task distribution is given, it may be possible to use it to resample instances from K and construct a more appropriate parameterized skill. A more general strategy is needed, however, in case the task distribution changes in a way that is not known to the agent.

CHAPTER 4

PARAMETERIZED MOTOR SKILLS ON A HUMANOID ROBOT

In this chapter we introduce a sample-efficient method for learning parameterized motor skills on a physical humanoid robot. Our method builds on the general framework introduced in Chapter 3 and extends it by introducing new mechanisms that allow for a more rapid skill acquisition when samples are expensive to obtain. Learning skills on a physical robot introduces significant challenges when compared to learning on a simulator, mainly because of the significant costs involved in practicing tasks, such as experimenter time, money, and robot wear and tear.

To address these challenges we develop a method for reusing seemingly unsuccessful policies as additional valid training samples for synthesizing a skill. This constitutes a simple type of counterfactual reasoning that allows an agent to use suboptimal policies—which may correspond to solutions to different problems—as additional training examples, thereby accelerating skill acquisition. We also show that simple rules can be created to automatically adjust the metaparameters of the models composing a skill, in particular those regulating the complexity of each chart model as a function of the number of samples available. This capability is important because it allows agents to adaptively control the expressiveness of their parameterized skill models as more samples are acquired.

We instantiate our learning framework via a set of specially designed regularized classification and multivariate regression models. We show that the resulting method is capable of constructing a *whole-body parameterized throwing skill, in a physical* *humanoid robot, from few training samples.* This demonstrates that our framework is not restricted to the realm of simulated problems, in which the cost of practicing tasks and evaluating candidate policies is typically negligible. Our experiments also confirm the qualitative findings presented in Chapter 3; in particular, that our learning algorithm can, from a small number of sample problems, predict appropriate solutions to novel tasks, and that it can autonomously learn the non-linear boundary separating low-dimensional charts. In our experiments, charts correspond to sub-skills that model the qualitatively different torso and arm movements required to execute different types of parameterized throws.

4.1 Skill Learning on a Robot

Our main objective is to demonstrate that parameterized skill learning can be successfully achieved on a physical robot. We assume the same optimization objective introduced in Chapter 3, but now consider the case when training samples are expensive to obtain, both in terms of robot wear and tear and experimenter time. As before, the acquisition of a parameterized skill starts with the construction of skill training set K. This set is constructed by drawing sample problems from P and learning an optimal or near-optimal policy for solving them. In this chapter we use a slightly different notation than the one used in Chapter 3. This allows for a simpler description of the regression and classification models used to instantiate our framework on a robot. Let $K \equiv \{(\tau_i, \theta_i)\}$, where τ_i is the i-th training task drawn from P and

$$\theta_i = (\theta_i(1), \ldots, \theta_i(M))^\top \in \mathbb{R}^M \tag{4.1}$$

are the M parameters of a policy for executing task τ_i. We follow the skill learning algorithm introduced in Section 3.3 and we use the set of sample tasks and corresponding policies to train a family of regression models mapping task parameters to policy parameters. Because policies for different subsets of T might be embedded in different lower-dimensional charts of the policy manifold, we once again first estimate the number D of such charts and only then train a separate regression model for each one. We employ a similar strategy to the one introduced in Section 3.4 and use ISOMAP to identify the global geometry of the policy manifold and the intrinsic number of non-linear degrees of freedom that underlie it. This allows the agent to estimate the number D of disjoint charts in the manifold, and to identify in which chart the policy of each training task is embedded. We organize this latter information into a training set $K_\chi \equiv \{(\tau_i, c_i)\}$, where τ_i is a training task and $c_i \in \{1, \dots, D\}$ identifies the chart to which task τ's policy belongs. We use K_χ to learn a classifier χ mapping the parameters of a task to the chart in which its policy is most likely embedded:

$$\chi : T \to \{1, \dots, D\}. \tag{4.2}$$

We learn χ via non-linear regularized logistic regression. The parameters optimizing this model are found via standard gradient descent.

In this instantiation of our framework we choose to represent parameterized skill functions Θ in terms of a set of functions $\Theta_c : T \to \mathbb{R}^M$, each one modeling one individual chart. Each Θ_c models how policy parameters change as we vary task parameters and is defined over all tasks whose policies are embedded in chart c. Let K_c be the set of all tasks which χ assigns to chart c: $K_c \equiv \{(\tau, \theta) \in K\}$ s.t. $\chi(\tau) = c$ and where π_θ is a policy for executing τ. We use each set K_c to construct the corresponding function Θ_c. In this chapter we represent each Θ_c as a ℓ^2-regularized linear regression model mapping *non-linear task features* to policy parameters. We then compose all learned chart models Θ_c to construct the overall parameterized skill function Θ:

$$\Theta(\tau) \equiv \Theta_c(\tau) \text{ s.t. } c = \chi(\tau) \tag{4.3}$$

and

$$\Theta_c(\tau) = \Big((\boldsymbol{w}_{K_c}^1)^\top \dots (\boldsymbol{w}_{K_c}^M)^\top \Big) \boldsymbol{\varphi}_{K_c}(\tau), \tag{4.4}$$

where $\boldsymbol{\varphi}_{K_c}(\tau)$ is an arbitrary V-dimensional vector of non-linear task features computed over τ, and $\boldsymbol{w}_{K_c}^j$, for each $j \in \{1, \dots, M\}$, is a V-dimensional vector given by

$$\boldsymbol{w}_{K_c}^j = \left(\Phi_{K_c}^\top \Phi_{K_c} + \lambda_{K_c} I \right)^{-1} \Phi_{K_c}^\top \boldsymbol{\Pi_j}. \tag{4.5}$$

Here, Φ_{K_c} is a $|K_c| \times V$ design matrix $\left(\boldsymbol{\varphi}_{K_c}(\tau_1)^\top \dots \boldsymbol{\varphi}_K(\tau_{K_c})^\top \right)$, λ_{K_c} is a regularization term and $\boldsymbol{\Pi_j}$ is a $|K_c|$-dimensional vector containing the j-th policy feature of each one of the $|K_c|$ training policies θ_i in K_c: $\boldsymbol{\Pi_j} = (\theta_1(j), \dots, \theta_{|K_c|}(j))^\top$. Note that Equation 4.4 is a multivariate regression model that implicitly assumes that policy features are approximately independent conditioned on the task. If this is known not to be the case, more expressive multivariate models that directly encode dependencies between task and policy features may be used.

Figure 4.1 depicts the overall process of executing a learned parameterized skill, which closely follows the general steps proposed in Chapter 3. First, a task τ is drawn from P; the classifier χ identifies the chart c in which that task's policy is most likely embedded; and finally, the regression model Θ_c associated with that chart is selected and maps the task parameters of τ to policy parameters $\theta_1, \dots, \theta_M$.

4.2 Sample Reuse

Consider the dart-throwing agent introduced in Chapter 3.4. It constructed parameterized skills by first learning policies for a small number of tasks and then generalizing them according to the algorithm presented in Section 3.3. Let us assume that this agent wishes, at some point during the skill acquisition process, to learn a policy for hitting target T_1. While learning to do so it may consider several candidate policies, most of which are unsuccessful at performing the task. Let us

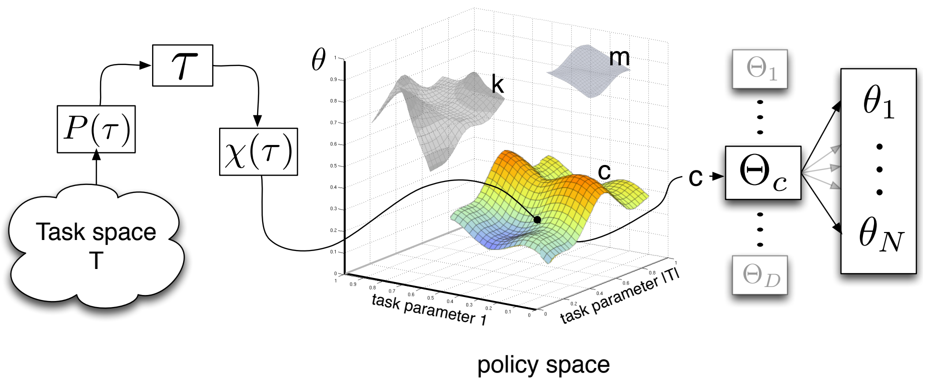

Figure 4.1. Steps involved in executing a learned parameterized skill composed of D chart models Θ_c, $c \in \{1, \ldots, D\}$.

assume that the agent evaluates a particular candidate policy which, when executed, hits an unintended target position T_2. Although it fails to hit the desired target, this policy may nonetheless be useful to the agent and should not be discarded. If T_2 or a nearby position ever need to be targeted, for instance, the agent can directly reuse that policy and avoid the cost of learning to hit T_2 from scratch. Instead of *directly storing* unsuccessful policies for later reuse, however, the agent may attempt to use them as *additional* training examples to learn a parameterized skill. The reuse of seemingly unsuccessful policies requires the agent to be capable of identifying which problems a given policy might solve. This corresponds to a type of counterfactual reasoning that allows an agent to use suboptimal policies—which may correspond to solutions to different problems—as valid training examples to construct a skill, thereby accelerating skill acquisition.

In the dart-throwing domain, a policy π_θ for hitting a target position τ is suboptimal if it hits a different target τ'. Even though it fails to solve τ, the unsuccessful policy π_θ can nonetheless be used to accelerate skill acquisition by incorporating the pair (τ', θ) into K, *without requiring the agent to ever explicitly train* for τ'.

Reusing unsuccessful policies as additional training samples accelerates the construction of the set K, the most time-intensive step in learning a parameterized skill.

This is possible because an agent practicing a single task analyzes a *sequence* of suboptimal policies, all of which may potentially be reused to solve novel problems without incurring any additional costs. Such sequences correspond to *trajectories* (in policy space) of potential problem solutions. An agent practicing a single task instance, therefore, can potentially acquire solutions to a large number of related problems, thereby decreasing the sample complexity of constructing the skill training set.

A first challenge to reuse policies is that it requires access to a function mapping from a policy, or observed effects of a policy (e.g., trajectories generated by it) to the parameters of the tasks in which that policy could be applied. This mapping can be seen as an *inverse model of the parameterized skill.* In some settings, such as the robotics domain introduced in the next section, this map can be easily handcrafted by an expert. Learning arbitrary inverse skill models from data, however, is significantly harder and beyond the scope of our work. In Section 6 we discuss possible future research directions related to this problem.

A second challenge to perform policy reuse it to determine when it is safe to do so; more specifically, in which situations an unsuccessful policy is guaranteed not to be an outlier with respect to a given chart model. This may occur because policy representations can be redundant and allow for different parameterizations, embedded in distinct charts, for solving a same problem. In this case, reusing policies may lead models Θ_c to be trained with incompatible solutions originating from different regions of the manifold. This is similar in nature to the problem (first discussed in Section 3.6) of attempting to learn inverse kinematic solutions by averaging examples on a non-convex set.

In order to ensure that a training set K_c does not contain samples from different charts, we design a simple rejection sampling strategy. Let τ' be the task unintentionally executed by a policy π_θ, while in search for a solution to τ. We add the tuple (τ', θ) to K_c, $c = \chi(\tau')$, iff

$$||\theta - \mathcal{P}_{K_c}(\theta)||_2 < \epsilon \ \mathrm{diam}(K_c), \tag{4.6}$$

where $\mathcal{P}_S(p)$ is the projection operator of p onto set S and $\mathrm{diam}(S)$ denotes the diameter of S, both of which are defined with respect to the ℓ^2-norm, and $\epsilon \in [0, 1]$ is a tunable parameter regulating how strict the rejection sampling strategy is. Intuitively, this condition tries to ensure that reused samples added to K_c are never too far away from the surface of that given chart.

4.3 The iCub Throwing Domain

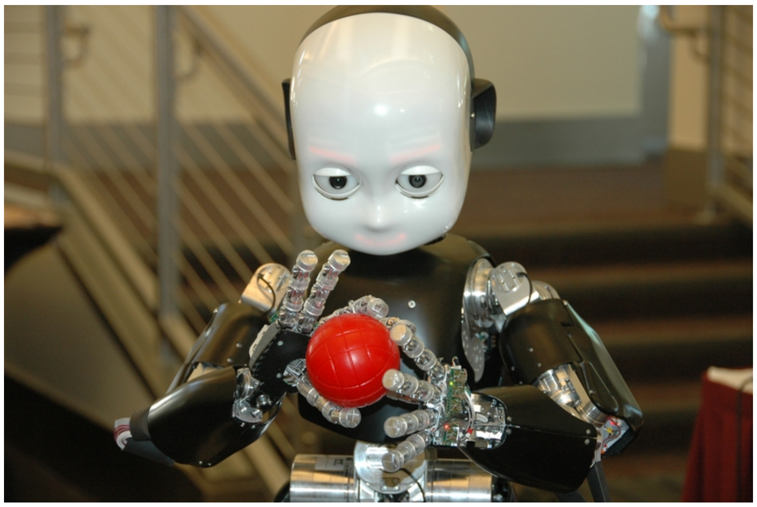

Figure 4.2. The iCub humanoid robot.

We evaluate the method introduced in Section 4.1 on an iCub robot tasked with learning to accurately throw plastic balls at parameterized target locations. The iCub is a humanoid robot with 53 actuated degrees of freedom built to have dimensions similar to that of a 3.5 year old child [48]. Our goal with this experiment is threefold: *1)* to test the effectiveness of our skill learning framework on a challenging physical system; *2)* to evaluate whether an effective whole-body parameterized throwing skill can be learned from few examples; and *3)* to evaluate whether the learning process can automatically identify and separately model the different sub-skills that may be required for executing qualitatively different types of tasks.

We place a 90cm $\times$ 90cm target board on the floor in front of the robot and allow targets (plastic bottles) to be placed anywhere on that board. The space T of tasks consists of a 2-dimensional Euclidean space containing all (x, y) coordinates at which targets can be placed. The performance of a policy (throw) corresponds to the distance between where the ball landed and the intended target.

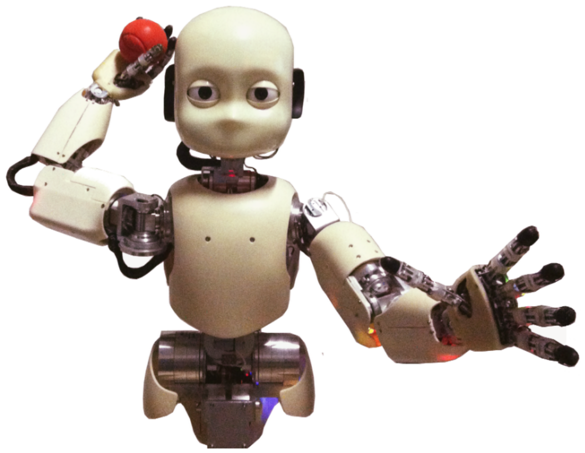

Figure 4.3. The iCub preparing for a throw.

Prior to acquiring a parameterized throwing skill we recorded the parameters of a *whole-body overhand throw* from kinesthetic demonstration. This throwing movement required simultaneously rotating the robot's shoulder, torso and forearm, pitching its torso and extending its elbow. Based on this recorded motion we defined a base policy for throwing whose parameters can be modified so that it hits different target positions. The base policy is a function of a 7-dimensional vector $(\theta_1, \ldots, \theta_7)^\top$ whose individual elements modify different aspects of a throw. Concretely, the parameters of the policy regulate the initial and final torso angles, torso rotation speed, torso pitch at the end of the movement, initial angle of rotation of the shoulder, amount of forearm rotation during the throw and total elbow extension. A throw initiates when the robot executes a predefined open-loop grasping motion to hold the plastic

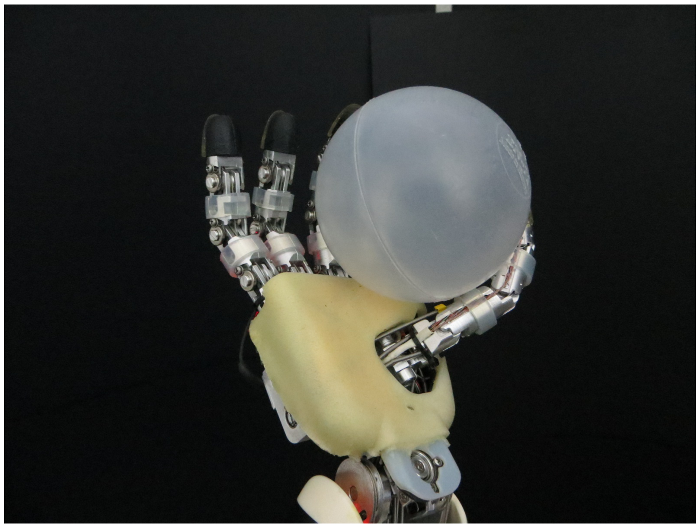

Figure 4.4. Grasp resulting from a predefined open-loop motion to hold the plastic ball.

ball (Figure 4.4). Executing a throw consists in using a PID controller to track the parameters of its corresponding policy. A throws also involves commanding the iCub's fingers to release the ball at a fixed movement during the movement. A sample throw is shown in Figure 4.5.

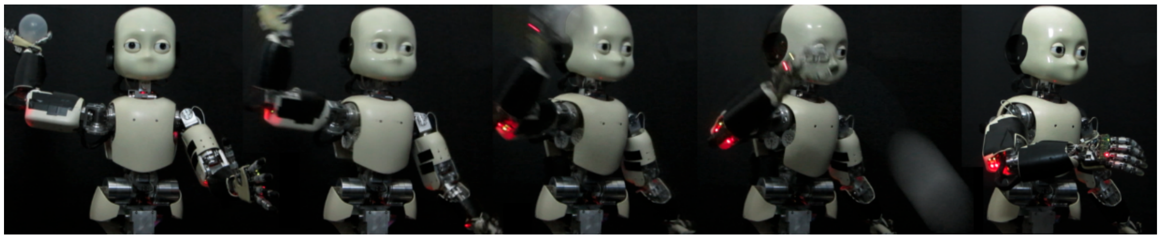

Figure 4.5. A sample throw executed by the iCub.

In order to fully instantiate our parameterized skill framework, we need to make two domain-dependent design decisions: *1)* what policy search algorithm to use when constructing the training set K; and *2)* what set of non-linear task features $\boldsymbol{\varphi}_{K_c}(\tau)$ to use when modeling the surface of each individual chart. The former decision influences the time to construct the skill training set, while the latter affects the expressiveness of each chart model Θ_c. We use the PI^2 algorithm [79] to learn individual policies.

PI^2 is policy search method derived from first principles of stochastic optimal control. We choose it because has been empirically shown to outperform standard gradient methods on robotics problems [70] and also due to its simplicity and low dependence on parameter-tuning. The stopping criteria used to determine when to terminate PI^2 corresponds to the moment when it finds a policy capable of hitting the desired target 4 out of 5 times. This criterion is needed to rule out noisy policy candidates. Finally, we construct the set of non-linear task features $\boldsymbol{\varphi}_{K_c}(\tau)$ used to model individual charts as a set of polynomial features computed over the original task representation. Because τ corresponds to the cartesian coordinates (x, y) of a given target, we define

$$\boldsymbol{\varphi}_{K_c}(\tau) \equiv \boldsymbol{\varphi}_{K_c}\big((x, y)\big) = \big(x^{e_1}, y^{e_2}, r^{e_3}, \alpha^{e_4}\big)^{\top}, \tag{4.7}$$

where $(r, \alpha) \equiv \big((x^2+y^2)^{\frac{1}{2}}, \text{atan2}(y, x)\big)$ is the polar-coordinate representation of (x, y) and each exponent $e_1, \ldots, e_4$ varies in the range $\{0, \ldots, P\}$, where P is the desired degree of the polynomial expansion.

We determined by cross-validation that an effective heuristic for choosing P is a logarithmic function of the number of training examples: $P = \lfloor \log_2(|K_c|) \rfloor$. Similarly, we determined that the regularization parameter λ_{K_c} can be effectively set by a linear function of the number of training examples: $\lambda_{K_c} = 0.1|K_c|$. These simple heuristic rules, even if only approximate, are extremely important in problems where samples are costly because they allow agents to adaptively control the expressiveness of their parameterized skill models as more training examples are acquired.

4.4 Experiments

Before discussing the performance of parameterized skill learning in this domain, we analyze and discuss important topological characteristics of the space of policies induced as we vary task parameters. We sampled 30 tasks (target positions) uniformly at random and placed targets (plastic bottles) at the corresponding locations. Near-optimal policies for executing each one of these tasks were learned using PI^2. We used ISOMAP to analyze these samples and estimate topological characteristics of the induced policy manifold. Our first important observation is that the residual variance of the produced quasi-isometric embeddings indicates that the intrinsic dimensionality of the skill manifold is two. This is expected since we are essentially parameterizing a 7-dimensional policy space by task parameters, which are drawn from a 2-dimensional space T. This implies that even though policies are part of a 7-dimensional space, they remain confined to 2-dimensional charts because there are only two degrees-of-freedom with which we can vary the tasks.

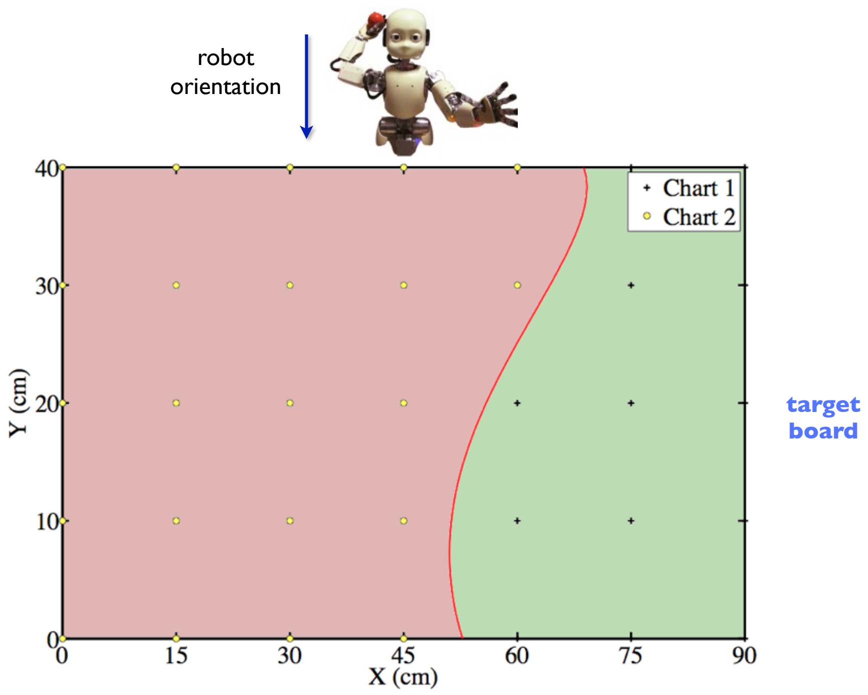

Figure 4.6. Target board positioned in front of the robot (not to scale). Targets in different regions of the board require the use of different specialized sub-skills.

Furthermore, our analysis of the manifold indicates that policies for different tasks are distributed over *two disjoint, but piecewise-smooth charts*. Figure 4.6 depicts the target board used in this domain and, overlaid on it, colors indicating which regions of the task space are associated to which chart. Policies for targets to the left of the robot are embedded in one of the charts, while policies for targets to the right of the robot lie on a second chart. The learned non-linear boundary separating these surfaces indicates the existence of two disjoint classes of policies, each one encoding a qualitatively distinct parameterized sub-skill (*throwing to the left; throwing to the right*) needed to execute different types of throws.

The identification of two classes of policy parameterizations is a demonstration of skill specialization: from a single root policy, learned from kinesthetic demonstration, two parameterized sub-skills are automatically identified and modeled. The need for these differentiated sub-skills partially reflects the dynamics of the robot, as well as physical constraints of its body and motor capabilities. The requirement for throws to be overhand and right-handed, for instance, implies that qualitatively different torso movements are needed in order to hit targets on the extreme left field of the robot.

Next, we analyze how the model of a low-dimensional chart encodes the way in which policy parameters vary as we smoothly vary a task. Figure 4.7 shows a representative subset of policy dimensions varying within a chart. The vertical axis on each inset figure corresponds to the value of a selected policy feature and the two other axes correspond to the task parameters. Our first important observation is that smooth task variations induce smooth, albeit non-linear, policy changes. Secondly, that even though the relation between task and policy parameters is non-linear and arguably difficult to be known *a priori*, there exists a clear structure that can be exploited—specifically, that policies for related tasks tend to be grouped on a same smoothly-varying, lower-dimensional chart.

We now discuss the performance of our parameterized skill learning method. Figure 4.8 shows the error in policy parameters predicted by the skill as a function of the number of examples used to train it. This is a measure of the relative error between the parameters predicted by Θ and parameters of a known solution for the same task. The lower the error, the closer the predicted policy is (with respect to the ℓ^2-norm)

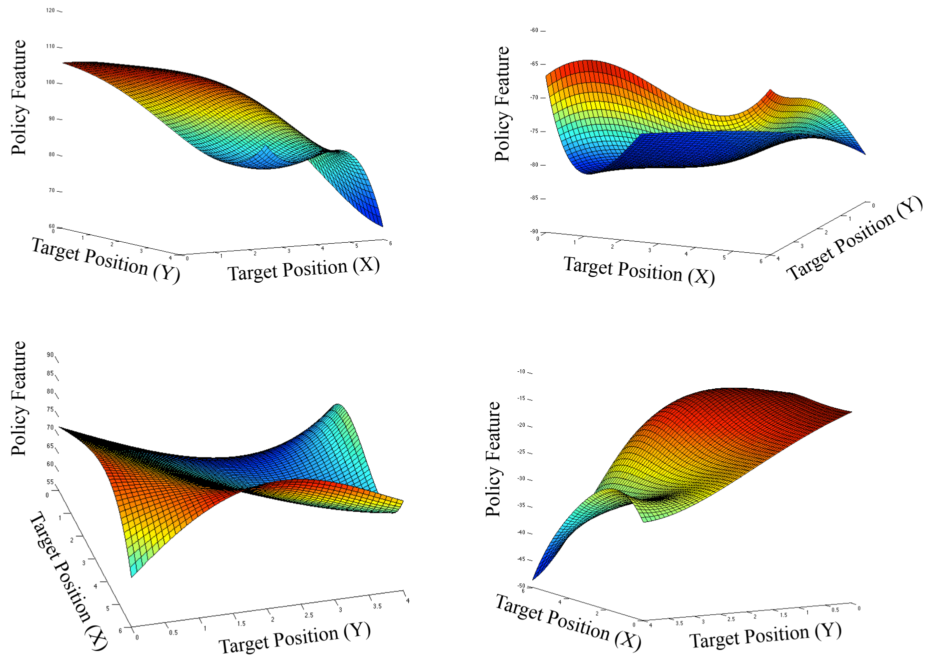

Figure 4.7. Examples of lower-dimensional projections of a learned chart Θ_c.

to a correct solution. In Figure 4.8 errors are averaged over 35 novel validation tasks sampled uniformly at random from the task space. After approximately 8 samples are presented to the parameterized skill, the average policy parameter error stabilizes around (but not at) zero. Obtaining this level of accuracy with few samples is only possible because the policy space is highly structured; specifically, because policies for similar tasks lie on low-dimensional charts whose regular topology can be exploited to better generalize known solutions to novel problems.

Because some policy representations might be sensitive to noise, high feature accuracy does not necessarily imply good policy performance. For this reason, we additionally measure the empirical effectiveness of predicted policies when directly applied to novel tasks. Specifically, we measure the average distance between the position where the ball hit and the intended target. In these experiments we defined a *minimum performance threshold*; this corresponds to the minimum acceptable performance of a predicted policy so that it can be considered sufficiently accurate. For

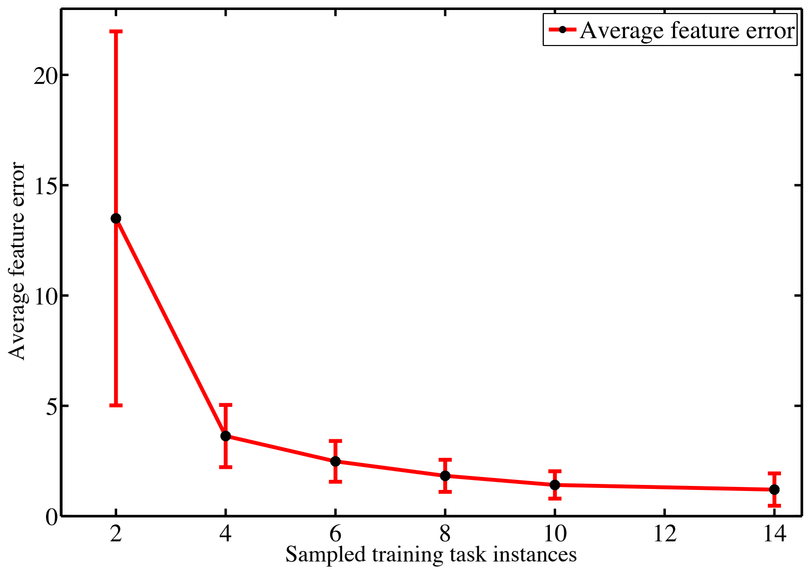

Figure 4.8. Average predicted policy parameter error as a function of the number of sampled training tasks.

this domain we set the performance threshold as the diameter of the plastic bottles used as targets: 6cm. This threshold was selected because throws that miss the intended target position by that amount are still capable, on average, of knocking down the plastic bottle. Figure 4.9 shows the average distance between where the ball hit and the intended target position as a function of the number of training samples. Distances were measured by directly executing a predicted policy and *before* any further policy improvement took place. Figure 4.9 shows, additionally, the expected performance of a baseline nearest-neighbors approach, which corresponds to selecting and directly executing the policy of the nearest known task in the training set K. Our experiments show that the generalization power of the parameterized skill allows its performance to strictly dominate that of the baseline approach. Furthermore, the learned parameterized skill produces throws meeting the specified performance threshold after around 8 training examples are presented. At this point, the robot is already capable of hitting arbitrary target regions of 6cm $\times$ 6cm within the 90cm $\times$

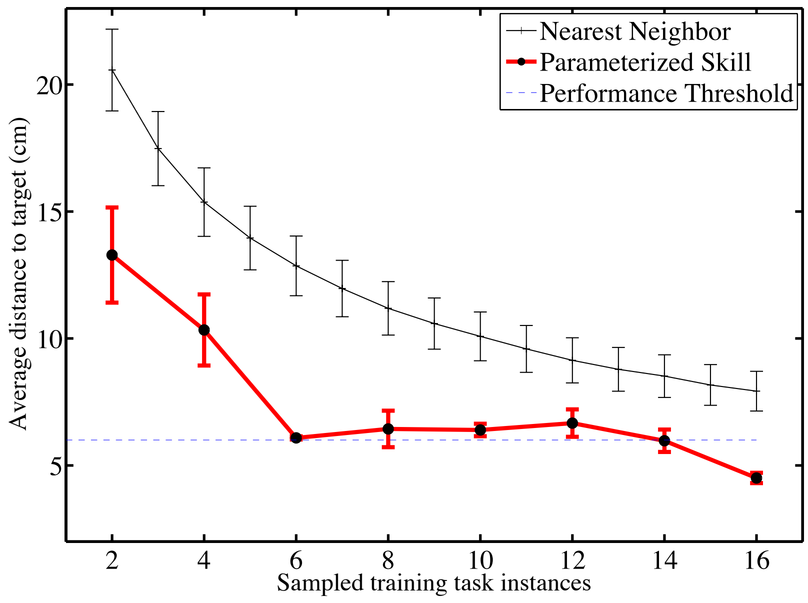

Figure 4.9. Average distance to target *(before any policy improvement)* as a function of the number of sampled training tasks.

90cm target board. As discussed in Section 3.5, policies predicted by the skill can be further improved via standard policy search methods if more accuracy is desired.

These experiments allow us to draw a few important conclusions: *1)* the base framework introduced in Chapter 3 can be augmented with a simple sample reuse mechanism and successfully applied to a challenging physical system in which samples are expensive to obtain; *2)* even though policy features cannot be perfectly predicted, an effective parameterized skill can be constructed from relatively few examples and used to predict accurate whole-body throwing policies; and *3)* our method is capable of exploiting the structure of the policy manifold to automatically identify and model the distinct sub-skills needed to execute different types of tasks. In particular, the charts identified by our method in this domain correspond to sub-skills that directly reflect important properties of the dynamics of the robot, as well as physical constraints of its body and motor capabilities.

4.5 Related Work

One alternative method for generalizing sample solutions to novel problems is to construct an abstract representation of a given set of related MDPs. Effective abstractions preserve the structural properties of an MDP and allow for a single abstract policy to be learned and used to solve new problem instances. Rajendran and Huber [55] introduce a method for constructing homomorphisms mapping a family of MDPs to a single abstract MDP. The authors assume that a set of MDPs encodes the same underlying *task type* if all of them achieve a *same equivalent goal state*. They introduce a method to learn an abstract representation of a family of structurally-equivalent MDPs (all of which achieving a *same goal*), thereby allowing for the construction of abstract policies that can be applied to similar problems. This objective is opposite to that of our method: we wish, by contrast, to construct a single skill capable of achieving *different goals*, given only a parameterized description of the goal.

Other techniques have been proposed to generalize solutions to novel problems. It is possible, for instance, to design flexible policy representations that allow the behavior of a robot to be parametrically adjusted in order to tackle novel motor tasks. Schaal et al. [59] introduced DMPs (see Section 2.2.5), a compact policy representation that allows for the re-parameterization of the initial and goal states of a movement. This policy representation cannot be used to encode arbitrary parameterized skills since the number and nature of their tunable parameters (or *metaparameters*) are fixed and pre-determined. Nemec et al. [50] partially addressed this limitation by extending the DMP framework with a gradient-based method to automatically determine appropriate metaparameters for executing a given task. Similarly, Kober et al. [33] proposed learning a mapping from task descriptions to DMP metaparameters. They assume that a single underlying motion primitive is sufficient to represent all tasks of interest and do not identify or model the qualitatively different strategies which may be needed to execute different tasks. Neumann et al. [51] explored a similar approach but their method requires learning a forward-model prior to training the skill. Finally, Stulp et al. [69] proposed further extensions to the DMP framework by allowing additional tunable parameters to be learned; they focus their attention, however, to the learning-from-demonstration setting.

4.6 Discussion

We have introduced a sample-efficient method for learning reusable skills on a physical humanoid robot. Our method instantiates the framework introduced in Chapter 3 via a set of specially designed models and extends it with a new mechanism for reusing seemingly unsuccessful policies. This constitutes a type of off-policy skill learning which reduces the number of samples needed to acquire a parameterized skill. Sample reuse of this kind, combined with simple heuristic rules to adaptively control the expressiveness of a skill model, allow for complex parameterized behaviors to be learned from very few examples.

We empirically show that our method is capable of identifying the low-dimensional charts where task policies are embedded. We also show that these charts correspond to qualitatively different sub-skills needed to execute different types of tasks. In our experiments on a physical robot, the need for differentiated sub-skills partially reflects the dynamics of the robot and physical constraints of its body. In particular, the requirement for overhand, right-handed throws results in the need for qualitatively different torso movements when hitting different target positions.

An important question not addressed in this chapter is that of how to select training tasks. Naïve task selection strategies draw tasks uniformly at random or directly from the target task distribution P. These sampling strategies, however, implicitly assume that all tasks are equally difficult and that the policy manifold has approximately uniform curvature. *Actively* selecting training tasks, by contrast, may allow a robot to more quickly uncover unknown parts of the policy manifold, thereby improving the overall readiness of the skill with fewer samples. This capability is particularly important in robotics domains, where exploration and sample collection are expensive. To address this problem we introduce an active skill learning technique in Chapter 5.

A second challenge not addressed in this chapter is that of learning inverse skill models. Reusing unsuccessful policies requires a mapping from the observed effects of a policy (e.g., trajectories generated by it) to the parameters of the tasks in which that policy could be applied. In some domains, such as the iCub throwing one, identifying these tasks is straightforward. In general, however, an *inverse parameterized skill model* needs to be learned—a mapping from policy parameters to tasks where they are applicable. Learning mappings of this kind allows for parameterized skills to be efficiently constructed even in domains where the correspondence between policy and tasks spaces is non-trivial. Learning an inverse skill model from data is a challenging problem with possible connections to Inverse Reinforcement Learning [52].

CHAPTER 5
ACTIVE LEARNING OF PARAMETERIZED SKILLS

In Chapters 3 and 4 we introduced methods for learning parameterized skills from a small number of training tasks. These methods constructed skills by practicing randomly selected tasks or by directly drawing them from the task distribution. These sampling strategies, however, do not necessarily result in rapid skill acquisition. In particular, they ignore which training tasks may improve skill performance the most and do not directly attempt to model their relative difficulties. It may be advantageous, therefore, for an agent to carefully select tasks for practice in order to guide the skill acquisition process and build on its strengths and mitigate its weaknesses.

In this chapter we introduce a framework for actively learning parameterized skills. This framework uses a novel task-selection mechanism capable of *identifying tasks that maximize expected skill performance improvement.* We derive the analytical expressions necessary to optimize it and propose a new spatiotemporal kernel tailored for non-stationary skill performance models. Our method is agnostic to policy and skill representation and scales independently of task dimension.

5.1 Setting and Motivation

In this chapter we adopt an optimization objective similar to the one introduced in Section 3.1, but now additionally assume that the skill learning agent can *actively choose* the tasks on which it wishes to train in order to more rapidly acquire a skill.

In the following sections we address the question of how such an agent, *given a distribution from which future tasks will be drawn*, should choose to practice. We posit that it should practice tasks that lead it to maximize skill performance improvement over all tasks in the distribution of interest. Intuitively, the tasks from which experience is most beneficial are those that allow the skill to better generalize to a wider range of related tasks. As we will show, identifying these tasks is not straightforward—sampling tasks according to the target distribution, for instance, is inefficient because it does not account for the varying difficulty of tasks. Furthermore, non-adaptive sampling strategies often ignore how some tasks may require policies qualitatively different from those of neighboring tasks, thus demanding more extensive training.

Consider, as a motivating example, a soccer-playing agent tasked with learning a parameterized kicking skill. This agent may wish to acquire the skill by practicing different types of kicks, such as accurate ones towards the goal or powerful but inaccurate kicks in the general direction of the opponent's half field. Policies for the former type of kick are harder to learn and generalize, since their parameters are very sensitive to the desired direction. Carefully choosing which kicks to practice allows the agent to identify which ones require less practice and which ones are more challenging, thus focusing on the aspects of the skill that can more readily be improved.

These observations are consistent with recent theories of how human experts acquire professional levels of achievement, which propose that skill improvement involves deliberate efforts to change particular aspects of performance [23]. Indeed, thoughtful and deliberate practice is one of the defining characteristics of expert performance in sports, arts and science: world-class athletes, for instance, carefully construct training regimens to build on their strengths and mitigate their weaknesses.

5.2 Active Learning of Parameterized Skills

A parameterized skill can be learned via a set of training tasks and their corresponding policies, as described in Chapters 3.3 and 4.1. The simplest strategy for constructing such a set is to select tasks uniformly at random or to draw them from the task distribution P. These strategies, however, ignore how a carefully-chosen task can improve performance not only on that task, but over a wider range of related tasks. Because parameterized skills naturally generalize policies to related tasks, it is advantageous for skill learning agents to focus not only on improving their performance at *single* tasks, but on the possible gains that practicing these tasks may bring to the overall skill.

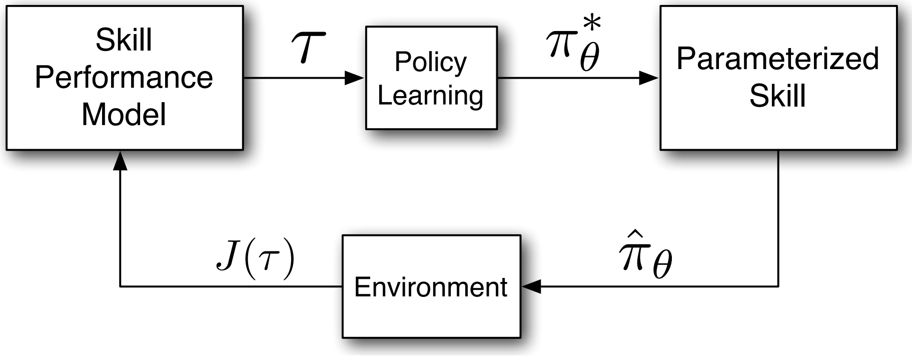

Figure 5.1. The process of actively learning a parameterized skill.

To address this challenge we introduce a framework for active task selection with arbitrary parameterized skill representations. This framework uses a Bayesian model of skill performance and a specially tailored acquisition function designed to *select training tasks that maximize expected skill performance improvement*. The proposed process for actively selecting training tasks consists of the following steps: *1)* identify, by means of a model of expected skill performance, the most promising task τ to practice; *2)* practice the task and learn a corresponding policy π_θ^*; *3)* update the parameterized skill; *4)* evaluate the empirical performance $J(\tau)$ of the updated skill at that task; and *5)* update the model of expected skill performance with the observed performance. These steps are repeated until the parameterized skill cannot be further improved or a maximum number of practicing episodes is reached. This training process is depicted in Figure 5.1.

5.3 A Bayesian Model of Skill Performance

Let $J\big(\pi_{\boldsymbol{\Theta}(\tau)}, \tau\big)$ be the performance achieved by a parameterized skill $\boldsymbol{\Theta}$ on task τ. To simplify notation we suppress the dependence on $\boldsymbol{\Theta}$ and write simply $J(\tau)$. We propose creating a Bayesian model capable of predicting skill performance over a set of tasks. This model allows an agent to predict skill performance—even at tasks it has never directly experienced—without incurring the costs of executing the skill. As will be shown in Section 5.4, this capability is particularly useful when estimating how different training tasks may contribute to improving the overall performance of a skill over a distribution of tasks.

Let $\mu(\tau)$ be predicted mean performance of the skill at task τ and $\sigma^2(\tau)$ the corresponding variance. One way of representing this predictive model is by using a Gaussian Process (GP) prior [56]. A GP is a (possibly infinite) set of random variables, any finite set of which is jointly Gaussian distributed. GPs extend Gaussian distributions to the case of infinite dimensionality, and thus can be seen as distributions over functions—in this case, predictive distributions over expected skill performance functions. To fully specify a GP one must define a mean function m and a positive-definite kernel k:

$$m(\tau) = E[J(\tau)]$$

$$k(\tau_p, \tau_q) = E\Big[\big(J(\tau_p) - m(\tau_p)\big)\big(J(\tau_q) - m(\tau_q)\big)^\top\Big].$$

A kernel specifies properties of J such as smoothness and periodicity. In this section and the following we do not make any assumptions about the form of k; in Section 5.5 we introduce a kernel designed specifically for use with our framework. We assume that the performance function J is zero-mean, so that m can be dropped from the equations; the extension for the non-zero mean case is straightforward.

Given k and a set of observations $D = \{(\tau_1, J(\tau_1)), \ldots, (\tau_N, J(\tau_N))\}$, both the log likelihood of the data conditioned on the model and the Gaussian posterior over skill performance, $P(J(\tau)|\tau, D)$, can be computed easily for any tasks τ. Under a Gaussian Process, the predictive posterior over skill performance at a task τ is normally-distributed according to $J(\tau) \sim \mathcal{N}(\mu(\tau), \sigma^2(\tau))$, where

$$\mu(\tau) = \mathbf{k}^\top \left(\mathbf{C_D} + \sigma^2 I\right)^{-1} \mathbf{y_D} \tag{5.1}$$

$$\sigma^2(\tau) = k(\tau, \tau) + \sigma^2 - \mathbf{k}^\top \left(\mathbf{C_D} + \sigma^2 I\right)^{-1} \mathbf{k}, \tag{5.2}$$

where $\mathbf{k} = (k(\tau, \tau_1), \ldots, k(\tau, \tau_N))^\top$, $\mathbf{C_D}$ is an $N \times N$ matrix with entries $(\mathbf{C_D})_{ij} = k(\tau_i, \tau_j)$, $\mathbf{y_D} = (J(\tau_1), \ldots, J(\tau_N))^\top$ and σ^2 is the additive noise we assume affects measurements of skill performance.

GPs are often used in Bayesian optimization to find the maximum of unknown, expensive-to-sample functions. To do so, a surrogate *acquisition function* is maximized. Acquisition functions guide the search for the optimum of a function by identifying the most promising points to sample. Standard acquisition functions include Lower Confidence Bounds, Maximum Probability of Improvement and Expected Improvement [64]. Although these criteria may seem, at first, appropriate for selecting training tasks, that is not the case: if applied to a skill performance function, standard acquisition criteria identify tasks on which the agent is expected to achieve a high level of performance. Our goal, by contrast, is to identify training tasks that result in the highest *expected improvement in skill performance.* This goal, formally defined in Equation 5.4, measures how much skill performance may improve *over a given target distribution of tasks* if the skill is updated by the practice of a selected task. A motivating principle behind this objective is that because parameterized skills naturally generalize policies to related tasks, an effective acquisition criterion should focus not on improving the performance at a single task, but on the possible performance gains that practicing it may bring to the skill as a whole.

In the next section we address this challenge by introducing a novel acquisition criterion specially tailored for parameterized skills. We derive closed-form expressions for this criterion and its gradient, thereby obtaining analytical expressions for the two quantities needed to optimize the process of selecting training tasks.

5.4 Active Selection of Training Tasks

An acquisition function to identify tasks that provide *maximum expected improvement in skill performance* should: *1)* take into account that tasks may occur with different probabilities and prioritize training accordingly; and *2)* model how the practice of a single task may, due to the generalization properties inherent to a parameterized skill, improve performance not only at that task but also over other similar tasks in T.

Let us assume we have practiced N tasks, $\tau_1, \ldots, \tau_N$, and used the corresponding (optimal or near-optimal) learned policies to construct a parameterized skill. Assume, furthermore, that we have evaluated the efficiency of the skill on tasks $\tau_1, \ldots, \tau_N$ and observed corresponding performances $J(\tau_1), \ldots, J(\tau_N)$. In what follows we annotate each training task τ_i with the time t_i it was practiced, which we denote by $\tau_i^{t_i}$. Here, time refers to the *order* in which tasks are practiced, so that the i-th task to have been practiced is annotated with time $t_i = i$. In the following sections we exclusively refer to these quantities as being *times* since that facilitates the intuitive interpretation of desired properties of our *time-changing*, non-stationary skill performance models.

Let D be a training set of tuples $\{\tau_i^{t_i}, J(\tau_i)\}$, for $i \in \{1, \ldots, N\}$. Given this training set, Equations 5.1 and 5.2 can be used to compute a posterior distribution over skill performance, $P(J(\tau)|\tau, D)$, for any tasks τ. Let J_t be the posterior distribution obtained when conditioning the Gaussian Process on all tasks practiced up to time t, and let $\mu_t(\tau)$ and $\sigma_t^2(\tau)$ be its mean and variance, respectively. We define *Skill Performance* (SP) as the expected performance of the skill with respect an arbitrary task distribution P:

$$SP_t = \int P(\tau)\mu_t(\tau)d\tau. \tag{5.3}$$

Furthermore, let the *Expected Improvement in Skill Performance* (EISP), given task τ, be the expected increase in skill performance if we assume that an agent practices τ, learns a policy for executing it with performance $j(\tau)$, and uses this experience to update its skill. Note, however, that unless the agent actually practices task τ (which generally incurs high training costs), it is not possible to know for sure the true performance that it might achieve on it by practice. The agent might choose, instead, to use its own model of skill performance to construct an *optimistic estimate* of the performance that it might achieve on a given task if it were to practice it. Let $j(\tau)$ be an *optimistic upper bound* on the performance that the agent may achieve at some task τ, computed with respect to its current posterior distribution J_t over expected performance. We formally define $j(\tau)$ in Equation 5.14 as an Upper Confidence Bound (UCB) on the agent's performance at τ.

To compute the EISP of a task τ we consider the Gaussian posterior, $\hat{J}_{t+1}$, that would result if J_t were to be updated with a new observation $(\tau, j(\tau))$. Let $\hat{\mu}_{t+1}(\tau)$ and $\hat{\sigma}^2_{t+1}(\tau)$ be the mean and variance of $\hat{J}_{t+1}$. The EISP of a task τ is defined as

$$\text{EISP}_t(\tau) = \int P(\tau')(\hat{\mu}_{t+1}(\tau') - \mu_t(\tau'))d\tau'. \tag{5.4}$$

EISP can be understood intuitively as a quantitative way of comparing tasks based on their likely contributions to improving the overall quality of a skill. Tasks whose practice may improve skill performance on a wide range of related tasks have higher EISP; conversely, tasks whose solutions are already well-modeled by the skill have lower EISP. Computing the EISP is similar to executing a mental evaluation of possible training outcomes: the agent uses its model of expected skill performance to estimate—without ever executing the skill—the effects that different training tasks may have on its competence across a distribution of problems.

We identify the training task τ^* that results in highest expected improvement in skill performance by computing the maximum of Equation 5.4. This corresponds to an acquisition function that selects training tasks according to:

$$\tau^* = \arg\max_{\tau} \mathrm{EISP}_t(\tau). \tag{5.5}$$

One way of evaluating Equation 5.5 is to use a gradient-based method. This requires an analytic expression for (or good approximation of) the gradient of $\mathrm{EISP}_t(\tau)$ with respect to arbitrary tasks. To make notation less cluttered, we now consider the case of 1-dimensional task parameters; the extension to higher-dimensions is straightforward. Assume, without loss of generality, that the parameter describing a task is drawn from a bounded interval $[A, B]$. To derive the expression for the gradient of EISP, we first observe that $\mu_t(\tau')$ in Equation 5.4 does not depend on τ and can be removed from the maximization. It is possible to show that the function to be maximized in Equation 5.5 is equivalent to:

$$\mathrm{EISP}_t(\tau) = G_t(\tau)^\top \Bigg(\big(\mathbf{C_t}(\tau) + \sigma^2 I\big)^{-1} \mathbf{y}(\tau) \Bigg), \tag{5.6}$$

where

$$\begin{aligned} G_t(\tau) &= \big(g_t(\tau_1^{t_1}),\ \ldots,\ g(\tau_N^{t_N}),\ g(\tau^{t+1})\big)^\top, & (5.7)\\ g_t(\tau_i^{t_i}) &= \int_A^B P(r) k(r^{t+1}, \tau_i^{t_i}) dr, & (5.8)\\ \mathbf{y}(\tau) &= \big(J(\tau_1), \ldots, J(\tau_N),\ j(\tau)\big)^\top, & (5.9) \end{aligned}$$

and where $\mathbf{C_t}(\tau)$ is the covariance matrix of the extended training set $D \cup \{(\tau^{t+1}, j(\tau))\}$:

$$\mathbf{C_t}(\tau) = \begin{pmatrix} k(\tau_1^{t_1}, \tau_1^{t_1}) & \dots & k(\tau_1^{t_1}, \tau^{t+1}) \\ \vdots & \ddots & \vdots \\ k(\tau_N^{t_N}, \tau_1^{t_1}) & \dots & k(\tau_N^{t_N}, \tau^{t+1}) \\ k(\tau^{t+1}, \tau_1^{t_1}) & \dots & k(\tau^{t+1}, \tau^{t+1}) \end{pmatrix}. \tag{5.10}$$

Furthermore, the gradient of $\text{EISP}_t(\tau)$ with respect to any given task τ is

$$\begin{aligned} \nabla_\tau \text{EISP}_t(\tau) &= \nabla_\tau G_t(\tau)^\top W_t(\tau) y(\tau) \\ &- G_t(\tau)^\top W_t(\tau) \nabla_\tau C_t(\tau) W_t(\tau) y(\tau) \\ &+ G_t(\tau)^\top W_t(\tau) \nabla_\tau y(\tau), \end{aligned} \tag{5.11}$$

where

$$\nabla_\tau G_t(\tau) = \Big(\underbrace{0, \ \dots, \ 0,}_{\text{N times}} \ \nabla_\tau g_t(\tau)\Big)^\top, \tag{5.12}$$

$$\nabla_\tau y(\tau) = \Big(\underbrace{0, \ \dots, \ 0,}_{\text{N times}} \ \nabla_\tau j(\tau)\Big)^\top, \tag{5.13}$$

and $W(\tau) = \big(C(\tau) + \sigma^2 I\big)^{-1}$. As previously discussed, $j(\tau)$ is as an optimistic upper bound on the performance that the agent might achieve at a task. We compute it by finding the upper endpoint of the 95% confidence interval around the mean predicted skill performance at τ:

$$j(\tau) = \mu_t(\tau) + 1.96\sqrt{\sigma_t^2(\tau)}.\ ^1 \tag{5.14}$$

[1]This upper bound is related to the one used to compute Upper Confidence Bound policies [41]. This latter approach, however, uses UCB bounds to identify *single* tasks with highest expected performance, while we use them as base values over which expected skill improvement on a task distribution is computed.

Then, $\nabla_\tau j(\tau) = \nabla_\tau\big(\mu_t(\tau) + 1.96\sqrt{\sigma_t^2(\tau)}\big)$. If we assume that the variance σ_t^2 of the process is approximately constant within infinitesimal neighborhoods of a given task, then $\nabla_\tau j(\tau) = \nabla_\tau \mu_t(\tau)$, which can be rewritten as

$$\begin{aligned}\nabla_\tau j(\tau) \quad &= \quad \Big(\nabla_\tau (k(\tau^t, \tau_1^{t_1}), \;\ldots,\; k(\tau^t, \tau_N^{t_N}))^\top\Big) \times \\ &\quad\quad \big(\mathbf{C_D} + \sigma^2 I\big)^{-1} \mathbf{y_D}. \end{aligned} \tag{5.15}$$

By rearranging the terms in Equation 5.11 it is possible to express $\nabla_\tau \text{EISP}_t(\tau)$ as a linear form $\phi_1 j(\tau) + \phi_2 \mathbf{y_D}$, where both ϕ_1 and ϕ_2 depend only on the kernel function k and on the task parameters sampled so far. This reveals an interesting property: given an arbitrary fixed set of training tasks, the gradient of EISP can be linearly decomposed into one component that depends solely on the performances y_D that are achievable by the skill, and another component that depends solely on the optimistic assumptions made when defining $j(\cdot)$. This implies that the direction of maximum improvement of EISP is *independently* influenced by *1)* the generalization capabilities of the skill—specifically, the actual performances it achieves on various tasks; and *2)* the optimistic assumptions regarding how further practice of a particular task may improve its performance.

Note that some of the equations in this section depend directly or indirectly on the choice of kernel k. In Section 5.5 we introduce a novel spatiotemporal kernel specially designed to better model skill performance functions, and in Appendix D we derive analytical expressions for the quantities involving it; namely, $\nabla_\tau k(\tau_i^{t_i}, \cdot)$, $g_t(\tau_i^{t_i})$ and $\nabla_\tau g_t(\tau_i^{t_i})$.

5.5 Modeling Non-Stationary Skill Performance Functions

Kernels encode assumptions about the function being modeled by a GP, such as its smoothness and periodicity. An implicit assumption made by standard kernels is that the underlying function is stationary—that is, it does not change with time. Kernel functions also specify a measure of similarity or correlation between input points, usually defined in terms of the coordinates of the inputs. If dealing with non-stationary functions, however, defining similarities is harder: when a point is resampled, for instance, we generally expect the similarity between its new and previous values to decrease with time. Note that the model of expected performance introduced in Section 5.3 is intrinsically non-stationary, since skill performance naturally improves with practice. If a standard kernel were to be used to model this function, outdated performance observations would contribute to the predicted mean, thus keeping the GP from properly tracking the changing performance.

To address this issue we introduce a new spatiotemporal kernel designed to better model non-stationary skill performance functions. Let us assume an arbitrary kernel $k_S(\tau_1, \tau_2)$ capable of measuring the similarity between tasks based solely on their respective parameters, τ_1 and τ_2. We expect k_S to be higher if comparing related tasks, and close to zero otherwise. Note that k_S does not account for the expected decrease in the similarity between observations of a task's performance at very different times. To address this issue we construct a composite spatiotemporal kernel k_C, based on k_S, capable of evaluating the similarity between tasks based on their parameters and on the times they were sampled. Let $k_C(\tau_1^{t_1}, \tau_2^{t_2})$ be such a kernel, where τ_i^t denotes a task τ_i sampled at time t. For k_C to be suitable for modeling non-stationary functions, it should ensure the following properties: *1)* related tasks have higher similarity if sampled at similar times; that is, $k_C(\tau_1^{t_1}, \tau_2^{t_2}) > k_C(\tau_1^{t_1+\Delta t}, \tau_2^{t_2})$, for small $\Delta t > 0$; *2)* if related tasks are sampled at significantly different times, no temporal correlation can be inferred and similarity is defined solely on their task parameters; that is, $k_C(\tau_1^{t_1}, \tau_2^{t_2}) \to k_S(\tau_1, \tau_2)$ as $|t_1 - t_2| \to \infty$; and *3)* the more unrelated tasks are, the smaller the correlation between them, independently of when they were sampled; that is, $k_C \to 0$ as $k_S \to 0$. The first property implies that if tasks are related, closer sampling times suggest higher correlation in observed task performance; the second property implies that nothing besides similarity in task space can be inferred if tasks are sampled at very different times; and the third property implies that sampling times, on their own, carry no correlation information if the tasks being compared are significantly different. To define k_C we introduce an isotropic exponential kernel $k_T(t_1, t_2)$ for measuring the similarity between sampling times:

$$k_T(t_1, t_2) = 1 + (C - 1) \exp\Bigg(- \rho^{-1}(t_1 - t_2)^2 \Bigg), \tag{5.16}$$

for some $C > 0$. k_T is such that $k_T \to C$ as $|t_1 - t_2| \to 0$, and $k_T \to 1$ as $|t_1 - t_2| \to \infty$. The parameter ρ is similar to the length-scale parameter in squared exponential kernels and regulates our prior assumption regarding how non-stationary the skill performance function is. We define the composite spatiotemporal kernel k_C as

$$\begin{aligned} k_C(\tau_1^{t_1}, \tau_2^{t_2}) &= k_S(\tau_1, \tau_2) \times \\ &\quad \Bigg(1 + (C - 1)e^{-\frac{(t_1 - t_2)^2}{\rho}}\Bigg). \end{aligned} \tag{5.17}$$

Intuitively, k_T boosts the correlation between tasks if they were sampled at similar times and ensures that only spatial correlation is taken into account as the difference between sampling times increases. Furthermore, note that when $C = 1$ all temporal information is ignored and k_C degenerates to the purely-spatial kernel k_S. Several methods, such as evidence maximization, are available to automatically identify suitable metaparameters for k_C and k_S [56].

Figure 5.2 depicts the predicted posterior mean and variance of a GP used to model a synthetic non-stationary function f. Two curves are shown: one for the predicted mean if using the purely-spatial kernel k_S, which does not take sampling time into account, and one for the improved predicted mean obtained if using the spatiotemporal kernel k_C. Lighter-colored points indicate older samples, while darker ones indicate more recent ones. Note how the latter kernel allows the predicted mean to correctly track the non-stationary function.

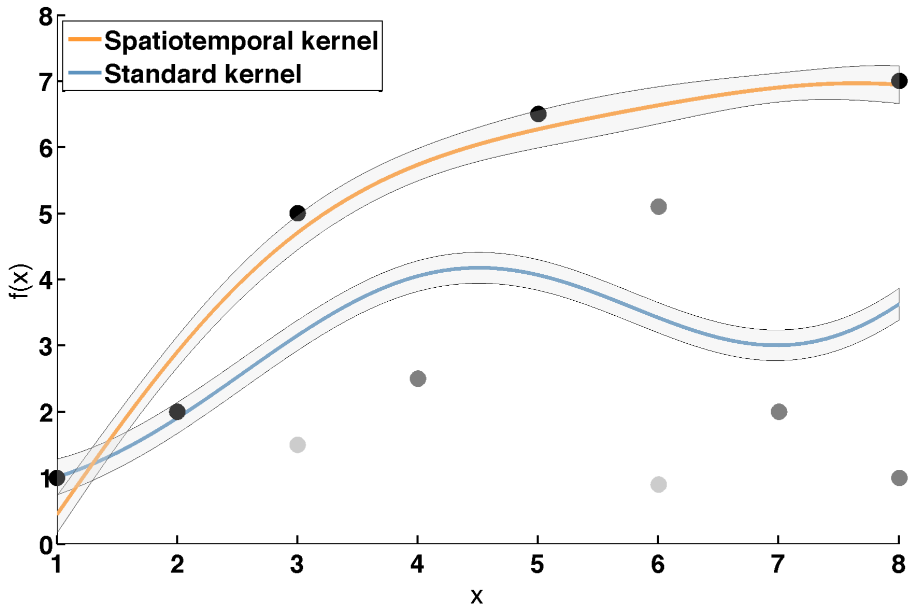

Figure 5.2. GP posteriors obtained when using a standard kernel and a spatiotemporal kernel to model a synthetic non-stationary function f. Lighter-colored points indicate older samples, while darker ones indicate more recent ones.

5.6 The Catapult Domain

We evaluate our method for active skill acquisition on a simulated catapult control problem where the agent is tasked with learning a parameterized skill for hitting targets on mountainous terrains (Figure 5.3). Targets can be placed anywhere on a 2-dimensional terrain with various elevations and slopes—both of which are unknown to the agent. The task space T consists of a single parameter describing the horizontal distance from the catapult to the target; note that this task parameterization does not convey any information about the elevation of the target or the geometry of the terrain, which makes the problem partially observable. Constructing a skill of this type is difficult because it requires generalizing over an ensemble of continuous-state, continuous-action control problems. In the following experiments we learn parameterized skills for controlling the catapult via Gaussian Process regression[2]. Parameterized skills in this domain map target positions to continuous launch parameters—namely, the angle and velocity with which a projectile is launched from the catapult. The performance of a launch is defined as the distance between where the projectile hits and the intended target. For more details about the equations governing this domain's dynamics, see Appendix C.

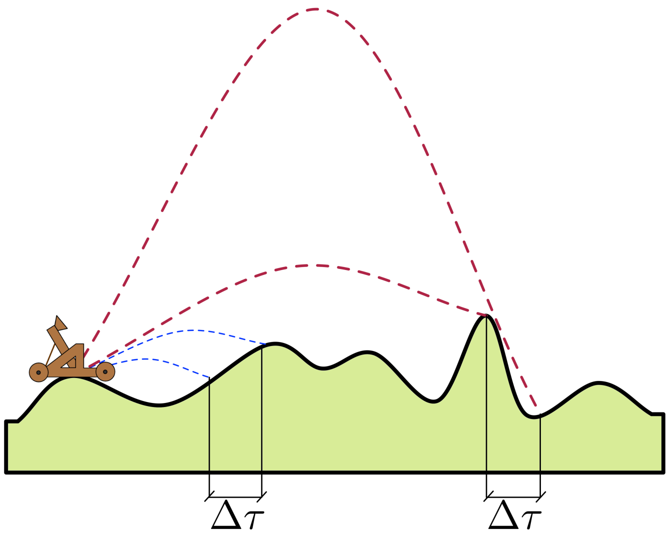

Figure 5.3. The Catapult Domain.

Determining which tasks to practice in this domain is challenging because irregular, non-smooth terrains may require significantly different launch profiles for hitting neighboring targets. Figure 5.3 shows how a change $\Delta\tau$ in task parameters may result in significantly different launch parameters depending on the region of the particular terrain being considered. Figure 5.4 depicts the policy manifold associated with launch parameters required for hitting various targets on a randomly-generated terrain (not shown). The 1-dimensional task space is represented by the red line, and

[2]This is similar to the approach taken by Kober et al. [33] but with no cost regularization. The GP-based parameterized skill model used here uses a squared exponential kernel—see Appendix D. In the following experiments, kernel parameters are found by evidence maximization.

gray lines mapping points in task space to policy space indicate policy predictions made by the skill. Discontinuities in this mapping indicate irregular regions of the policy manifold in which generalization is difficult. Finally, note that identifying the target task with maximum EISP corresponds to optimizing a one-step look-ahead sampling strategy to quickly uncover the structure of this manifold.

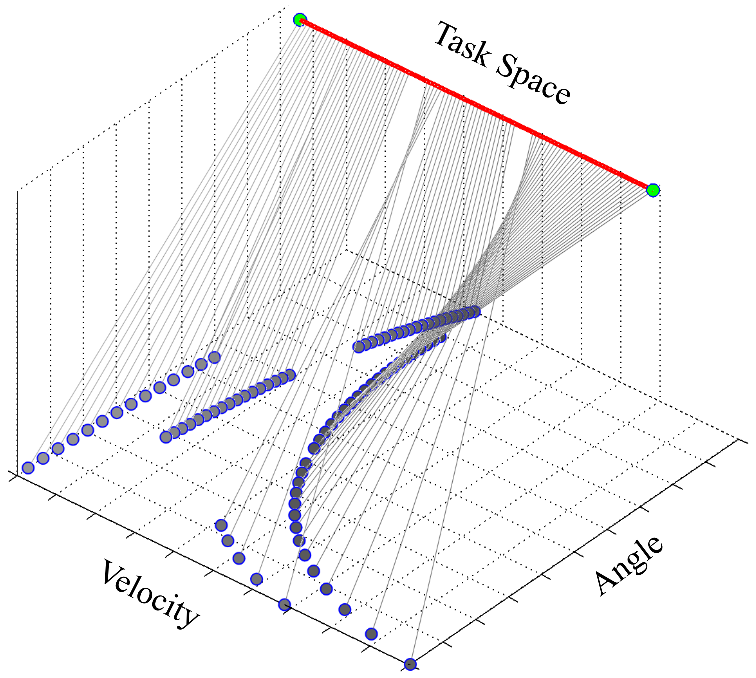

Figure 5.4. Policy manifold of the catapult domain. The red line represents the task space. Gray lines connecting task to points in the policy space indicate predictions made by the skill. Discontinuities in the mapping indicate task regions where generalization is difficult.

We compare the performance of our method with four alternative approaches. Two of them are baseline, non-adaptive sampling strategies: selecting tasks uniformly at random, and probabilistically according to the task distribution P. We also compare with two active acquisition criteria commonly used in Bayesian optimization: Expected Improvement (EI) and Lower Confidence Bound (LCB). Figures 5.5 and 5.6 show skill performance as a function of the number of tasks practiced, for different task selection methods. In these experiments, skill performance was measured by evaluating the skill on a set of novel tasks; the observed performances were weighted by the task distribution P to reflect whether the agent was competent at tasks of higher interest. To report an absolute measure of skill quality we computed the mean squared difference between overall performance of the learned skill and the maximum performance that can be achieved by the skill model. The lower the difference, the better the skill is at solving tasks from the distribution of interest. All curves are averages over 50 randomly generated terrains.

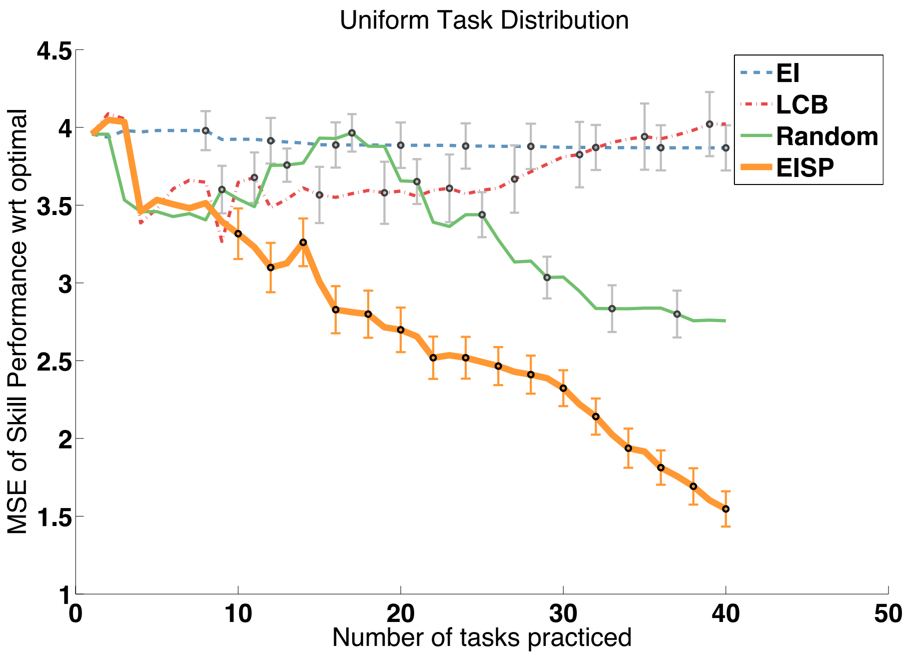

Figure 5.5. Average skill performance as a function of the number of sampled training tasks (uniform task distribution).

Figure 5.5 shows how skill performance changes as the agent practices more tasks, assuming a uniform target distribution P of tasks. Note that in this case both non-adaptive sampling strategies—i.e., selecting tasks at random or drawing them from P—are equivalent. Similarly, Figure 5.6 shows skill performance as a function of tasks practiced but for the case of a non-uniform target distribution P—that is, an agent with stronger preference for becoming competent at targets in particular regions of the terrain. Here, P was defined as a Gaussian centered at the midpoint of the terrain.

Under both types of task distribution, EI performed worse than all other methods. EI selects tasks whose individual performances are expected to improve the most by practice. This criterion leads the agent to repeatedly practice tasks that are already well-modeled by the skill but which may be marginally improved. This causes the agent to ignore regions of the task space in which it is not yet competent. LCB suffers from a similar shortcoming; it selects tasks in which the skill has *lowest* expected performance, thus focusing on improving the agent's weaknesses. This often leads the agent to obsessively practice tasks that it may be unable to execute well. Finally, both random selection of tasks and selection according to the target distribution P fail to account for the varying difficulty of tasks. These criteria choose to practice problems independently of the skill's current capability of performing them; furthermore, they often practice tasks that are irrelevant according to the target task distribution. EISP, on the other hand, correctly identifies which tasks can improve skill performance the most, and takes into account both their relative difficulties and how well their solutions generalize to related tasks.

Figure 5.7 depicts the probability with which different methods explore different regions of the task space. The horizontal axis represents the possible tasks the agent may choose to practice; in this domain, these correspond to the different positions within a terrain where targets may be placed. The vertical axis represents the probability with which each method practices on specific regions of the terrain. Figure 5.7 shows results for a randomly-generated sample terrain (not shown); similar qualitative results were observed in other terrains. This experiment allows us to draw a few important conclusions. Random sampling and sampling according to the task distribution P do not adapt their task-selection strategies according to the available training tasks or based on the current performance or generalization power of the skill. EI identifies three regions of the task space in which the skill is effective and focuses on trying to further improve those. LCB, in contrast, samples more densely

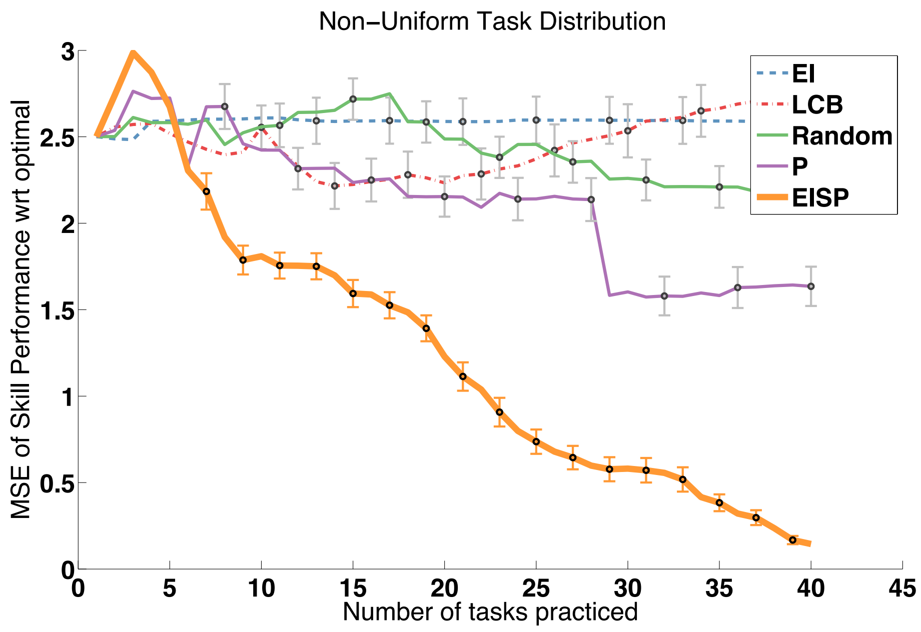

Figure 5.6. Average skill performance as a function of the number of sampled training tasks (non-uniform task distribution).

regions that contain difficult tasks. However, because it does not model whether skill performance is expected to improve, it often focuses on tasks that are too difficult. EISP prioritizes practice according to the target distribution P *and* selects problems according to how much they are expected to contribute to improving skill performance. In particular, note how it chooses to practice less on tasks at the beginning of the task range, even though those tasks have a high probability of occurring according to P. This happens because EISP quickly realizes that solutions to those tasks can be easily generalized and that no further samples are required. Finally, the peak of samples collected by EISP at the end of the task range corresponds to a particularly difficult part of the terrain which requires prolonged practice. Note how EISP devotes less attention to that region than EI since it is capable of predicting when no further skill improvement is expected.

We can draw a few important conclusions from our results: *1)* non-adaptive strategies implicitly assume that all tasks can be equally well generalized by the skill—or,

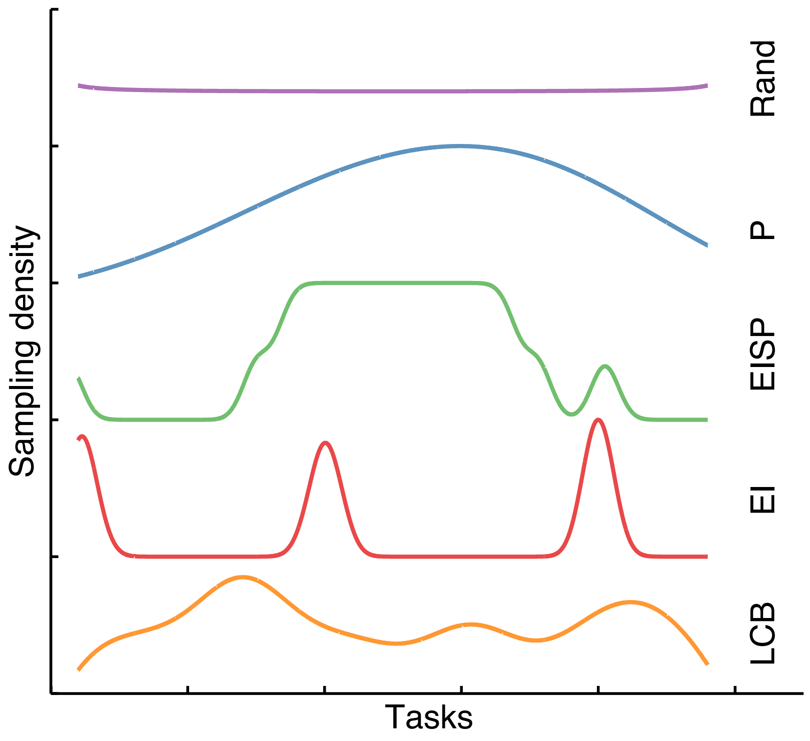

Figure 5.7. Density of samples collected by different training strategies.

equivalently, that the manifold has approximately uniform curvature; this causes them to unnecessarily practice tasks that may already be well-modeled by the skill; *2)* the LCB criterion always selects tasks with lowest predicted performance, often repeatedly practicing poorly-performing tasks as long as they can be infinitesimally improved; *3)* EI focuses on further improving single tasks at which the agent may already be competent, thus refraining from practicing more difficult ones, or ones that are more important according to the task distribution; and finally *4)* because EISP uses a model of expected performance to infer the generalization capabilities of a skill, it correctly identifies the task regions in which practice leads to better generalization across a wider range of tasks. Furthermore, the use of an expected skill performance model allows for the identification of tasks that are either too difficult to execute or whose performance cannot yet be further improved, thus leading the agent to focus on problems that are compatible with its current level of competence.

5.7 Related Work

In Section 5.5 we introduced a spatiotemporal composite kernel to model non-stationary skill performance functions. Other methods have been proposed to allow GP regression of non-stationary functions. Rottmann and Burgard. [57] proposed to manually assign higher noise levels to older samples, causing them to contribute less to the predicted mean. This approach relies on domain-dependent cost functions, which are difficult to design. Spatiotemporal covariance functions, defined similarly to our composite kernel k_C, have been proposed to solve recursive least-squares problems on non-stationary domains. These functions, constructed by the product of a standard kernel and an a Ornstein-Uhlenbeck temporal kernel, often require estimating a forgetting factor [82].

Previous research has also addressed the problem of selecting training tasks to efficiently learn a skill. Hausknecht and Stone [26] constructed a skill by solving a large number of tasks uniformly drawn from the task space. They exhaustively varied policy parameters and identified which tasks were executed by the resulting policies, thus implicitly acquiring the skill by sampling *all* possible tasks. Kober, Wilhelm, Oztop, and Peters [33] proposed a cost-regularized kernel regression method for learning a skill but did not address how to select training tasks. In their experiments, tasks were sampled uniformly at random from the task space. Similarly, da Silva et al. [17] proposed to acquire a skill by analyzing the structure of the underlying policy manifold, but assumed that tasks were selected uniformly at random. Finally, Baranes and Oudeyer [2] proposed an active learning framework for acquiring parameterized skills based on competence progress. Their approach is similar to ours in that promising tasks are identified via an adaptive mechanism based on expected performance. However, their approach, unlike ours, requires a discrete number of tasks and can only optimize the task-selection problem over finite and discrete subsets of the task space.

The overall question of how to actively select training tasks is also related to the idea of intrinsic motivation. Intrinsic motivation leads animals—even in the absence of explicit rewards or reinforcement—to engage in exploration, play and other curiosity-driven behaviors. The notion of self-motived learning and curiosity has been studied both in psychology (e.g., by Harlow [24]) and in the machine learning and developmental robotics communities [60, 61, 67, 65]. Psychologists distinguish between extrinsic motivation, which means doing something because of some specific rewarding outcome, and intrinsic motivation, which refers to doing something because it is inherently "interesting or enjoyable" [58]. In a classic paper, White [85] argued that intrinsically motivated behavior is essential for an organism to gain the competence necessary for autonomy, where autonomy refers to the extent to which an organism is able to achieve mastery over its environment. An agent engaged in actively learning skills can be seen as intrinsically motivated to improve its mastery of the skill.

The idea of intrinsic motivation has also been extensively studied in the machine learning community. We highlight, in particular, the pioneering work of Barto et al. (e.g., [5, 63, 4], where intrinsic motivation signals are used (for the first time, to the best of our knowledge) to autonomously identify and construct a set of skills. Also of importance is the work of Schmidhuber (e.g., [60, 61]), where he posits that intrinsically motivated agents are interested in learnable but yet unknown regularities, but bored by predictable or inherently unpredictable events. Schmidhuber introduced a series of computational models based on the notion of *compression progress*, where compression generally refers to the acquisition of more compact predictive models. An agent maximizing compression progress might, for instance, be driven to observe particular aspects of the world that allow it to rapidly acquire more compact models of the environment. The idea of competence progress has also been explored by others. Stout et al. [66] introduced a method to choose, at any given moment, which of a number of skills (or options) an agent should attempt to improve. Their method differs from ours in that it aims at efficiently learning a repertoire of options for solving independent problems. They do not, however, address the case in which attempting to improve one option might be beneficial because that option's policy can be generalized and used to solve other related problems.

5.8 Discussion

We have introduced a framework for actively learning parameterized skills. Our method uses a novel acquisition criterion capable of identifying tasks that maximize expected skill performance improvement. We have derived the analytical expressions necessary for optimizing it and proposed a new spatiotemporal kernel especially tailored for non-stationary performance models. Our method is agnostic to policy and skill representation and can be coupled with any of the recently-proposed parameterized skill learning algorithms [33, 17, 51, 19].

This work can be extended in several important directions. The composite kernel k_C can be used to compute a posterior over expected *future* skill performance, which suggests an extension of EISP to the case of multistep decisions. This can be done by evaluating the predicted posterior mean (Equation 5.1) over a set of test points $\{\tau_i, T + \Delta\}$, where T is the current time and Δ is a positive time increment. This is useful in domains where a one-step look-ahead strategy, like the one optimized in Equation 5.5, is too myopic.[3] Secondly, our model uses a homoscedastic GP prior, which assumes constant observation noise throughout the input domain. This may be limiting if the agent has sensors with variable accuracy depending on the task—for instance, it may be unable to accurately identify the position of distant targets. Heteroscedastic GP models, such as the ones proposed by Kuindersma et al. [42], may

[3]A myopic strategy might fail to realize the long-term impact of practicing specific tasks. Consider, for instance, that even if two tasks have similar potential for improving skill performance, one of them—if trained first—may facilitate learning of other tasks in the future.

be used to address this limitation. Finally, taking advantage of human demonstrations might help biasing EISP towards tasks which an expert deems relevant, which suggests an integration with active learning from demonstration techniques [62].

CHAPTER 6
FUTURE WORK

The methods introduced in this thesis can be extended in several important directions. Besides the ones already discussed in each individual chapter, we highlight six important future research directions:

6.1 Extending EISP to a Fully Bayesian Formulation

The EISP formulation introduced in Equation 5.4 assumes that the posterior distribution over performance, $\hat{J}_{t+1}$, is computed by updating the current distribution J_t with a synthetic observation $(\tau, j(\tau))$, where $j(\tau)$ is an optimistic upper bound on performance. Instead of assuming a single optimistic upper bound, a fully Bayesian formulation of EISP is also possible. This formulation computes the expected improvement in skill performance by considering all possible observations of performance after training a task τ:

$$\text{EISP}^*(\tau) \;=\; \int \phi(p; \mu(\tau), \sigma^2(\tau)) \bigg(\int P(\tau')(\hat{\mu}^p_{t+1}(\tau') - \mu_t(\tau'))d\tau' \bigg) dj \tag{6.1}$$

where $\hat{\mu}^p_{t+1}(\cdot)$ is the mean of the posterior distribution $\hat{J}_{t+1}$ resulting from updating J_t with an observation (τ, p), and $\phi(p; \mu, \sigma^2)$ is the PDF of the Normal distribution with mean μ and variance σ^2.

It is not clear, however, if it is possible to efficiently evaluate this alternative formulation of EISP. Further study is also necessary to determine whether closed-form solutions, such as the ones presented in Section 5.4, are possible under this new formulation.

6.2 Efficiently Computing EISP by Incrementally Updating Concentration Matrices

Computing EISP given a task involves executing a matrix inversion operation. When evaluating Equations 5.6 and 5.11 for various candidate tasks τ, the covariance matrix C_D (Equation 5.2) is updated by adding one row and one column to it. This results in an extended matrix $\mathbf{C_t}(\tau)$, which then needs to be inverted. Because both of these matrices are related, however, it is possible to use the Bordering Method for inverting block matrices and obtain an incremental update for the inverse of $\mathbf{C_t}(\tau)$. This simple modification is important if our method is to be applied in real-time systems where the decision of which tasks to practice is time-bound.

6.3 A Mechanism for Merging Redundant Sub-skills

An agent practicing a task might choose to use a parameterized skill to obtain an initial guess for that task's policy. This might introduce biases regarding which policies are considered during the learning process, and may ultimately affect how many charts or sub-skills are discovered. A stable skill acquisition scheme should be able to identify, and remove or merge, spurious sub-skills resulting from biases introduced by bootstrapping task policies.

As an example of the type of bias we refer to, assume that two given tasks, τ_1 and τ_2, can be executed by policies π_1 and π_2, respectively. Furthermore, assume that the parameters of π_1 can be smoothly modified by a learning algorithm to obtain solutions to a large number of related tasks, including τ_2. Assume, by contrast, that π_2 *cannot* be similarly modified to generate solutions to other tasks. If the agent chooses to practice τ_1 first, the skill, when queried about a solution to τ_2, will produce a policy biased towards π_1. Because π_1 can be smoothly modified to generate a solution to τ_2, both learned policies remain near in policy space and the manifold analysis reveals a single chart. If the agent chooses to practices τ_2 first, however, the skill, when queried about a solution to τ_1, will produce a policy biased towards π_2. Because we assume that this policy *cannot* be easily modified to generate solutions to other tasks, the learning algorithm will eventually converge to an alternative solution for τ_1 which may be far (in policy space) from τ_2. Since these policies are not neighbors in the space of policies, the analysis of the manifold will most likely identify multiple charts.

To mitigate the impact of potentially spurious chart models, a chart or sub-skill merging mechanism is needed. Reducing the number of redundant or unnecessary sub-skills is important because a larger number of models implies that more samples are needed to train the skill. One possible approach to address this issue is to identify when two sub-skills can be interchangeably used to execute, with similar performance levels, a large number of tasks. The agent might then use this information to remove or merge redundant sub-skills.

6.4 Smooth Chaining of Parameterized Skills

Assume that an agent has acquired two parameterized skills, A and B. Is it possible to use these skills to derive a new *joint skill*, $\widehat{AB}$, parameterized by the task parameters of A and B, and capable of producing behaviors that correspond to the smooth sequential execution of A and B? As an example, consider a soccer-playing agent that has learned a sprinting skill, parameterized by desired running speed, and a kicking skill, parameterized by desired kicking force. If the agent learns these skills independently, there are no guarantees that the sprinting skill will, after termination, leave the agent in a state from which the kicking skill can be immediately executed.

One way to construct the joint skill $\widehat{AB}$ is to *1)* explicitly model or learn the initiation and termination sets of A and B; *2)* redefine A so that it has an additional task parameter, g_A, corresponding to the desired termination state of that skill; and *3)* execute $\widehat{AB}$ by temporally sequencing A and B, but enforcing that g_A is set of some state in the intersection of A's termination set and B's initiation set. This guarantees, by construction, that the two skills will be smoothly executed because A terminates in a state from which B can be immediately invoked. However, a mechanism for smoothly combining A and B without having to redefine (and retrain) A over a larger task space would be preferable.

6.5 Learning Reusable Skills Under Heterogenous State and Action Spaces

A parameterized skill is a function capable of predicting appropriate parameters θ of a policy $\pi_\theta : S \times A \to [0, 1]$ so that it solves some given problem τ. Because the domain of the policy being parameterized is fixed ($S \times A$), all MDPs that might be solved by the skill need share the same state and action spaces. It would be beneficial to extend our framework in a way that allowed skills to generalize over problems with different state and action representations. Konidaris et al. [36, 40] introduced a method for learning reusable options by representing their policies not in the original state space, but in an *agent-centered* feature space. This approach assumes that the set of shared features defining the agent-centered space is known *a priori*. A mechanism for autonomously discovering the common space of shared task features would significantly extend the applicability of our framework.

6.6 Learning Inverse Parameterized Skill Models

Reusing seemingly unsuccessful policies as additional training samples (Section 4.3) requires a mapping from observed effects of a policy (e.g., trajectories) to tasks in which that policy could be used. In some problems this information can be readily extracted from trajectories, such as those in which the agent attempts to reach a given goal state. Let an inverse parameterized skill model Θ^{-1} be a function mapping *trajectories* to tasks:

$$\Theta^{-1} : \xi \to T$$

where ξ is a trajectory in the form $(s_0, a_0, r_0), \ldots, (s_n, a_n, r_n)$, s_i, a_i and r_i are, respectively, the state, action and instantaneous reward at time i, and T is the continuous space of tasks.

Consider the dart-throwing agent introduced in Section 3.4, and assume that it continually observes the location of the dart after a throw. If episodes end whenever the dart hits a wall, the last state of any trajectory will contain the coordinates where the dart hit. An inverse skill model would, in this case, map any given trajectory ξ of length n to the last state in it: $\Theta^{-1}(\xi) = s_n$. More generally, however, Θ^{-1} might depend on the entire trajectory instead of only on its last state.

It may be also possible to manually construct an inverse model Θ^{-1} in problems where the reward function has a specific type of internal structure. Assume that the reward function can be decomposed into one term that is *invariant with respect to the task* and one that is not. Then, the performance of the agent at a task τ, given the trajectory ξ, is

$$J(\xi) = R(\xi) + G(\xi, \tau), \tag{6.2}$$

where R is the component of the reward function that is invariant to task and G is the one that depends both on the task and on the observed trajectory. Consider a basketball-playing agent tasked with learning a skill for shooting a ball through a basket. Assume that the basket is at coordinates $(0, 0)$ and that the skill is parameterized by the agent's own position (x, y) and orientation α on the court. Assume that policies are open-loop throwing motions that depend on (x, y, α). Assume, furthermore, that the reward function is composed of a constant penalty term $-\epsilon$ for every time step before the throw and a penalty $-d$ received at the end of the episode, where d is the distance between where the ball landed and the basket. The performance of a throw is

$$J(\xi) = -n\epsilon - d$$

where ξ is a trajectory of length n. Note that J is in the form of Equation 6.2. By knowing the structure of the reward function it is possible to estimate how far ahead of the agent (distance D) a given policy throws the basketball—see Figure 6.1.

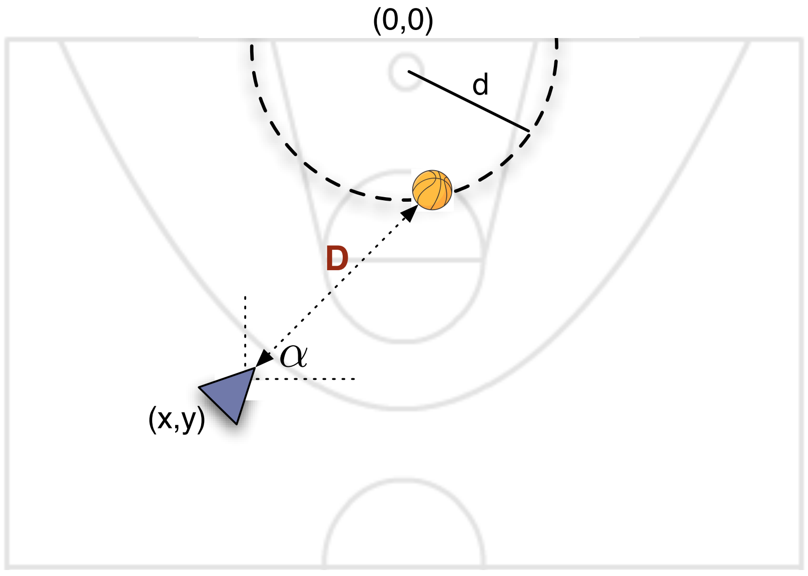

Figure 6.1. Inferring tasks where a policy is applicable given the observed return of a trajectory.

The domain described above has one important invariant that can be exploited; namely, that throws have the same effect on the ball independent of where the player is on the court. It is possible to use this observation in conjunction with the inferred distance D to identify other tasks where the current policy could be applied. Specifically, the agent can use the current policy to execute any other tasks $\tau = (x', y', \alpha')$

where $||(x', y')|| = D$ and $\alpha' = \arcsin(\frac{y'}{D})$. Here, α' is the angle the agent needs to reorient itself to so that it faces the basket. These task parameters put the agent at the correct distance from the basket. If the agent can freely rotate its body to any desired orientation, the current policy can also be applied to any tasks $\tau = (x', y', \omega)$ where $||(x', y')|| = D$ and ω is the angle required so that the agent faces the basket.

We have so far discussed cases in which Θ^{-1} can be manually specified by an expert with prior knowledge about invariants of the domain, as well of the structure of the reward function. In general, however, Θ^{-1} may need to be learned from data.

We believe that learning a suitable Θ^{-1} entails *finding the task with maximum reward for a given trajectory.* This optimization problem is related to that of Inverse Reinforcement Learning (IRL), where the goal is to identify a reward function under which an optimal policy matches sample trajectories provided by an expert [52, 43]. One important difference is that identifying a suitable reward function does not necessarily help to identify the tasks that maximize it.

Finally, note that not all parameterized skills benefit from sample reuse. Consider an agent learning to balance a pole under different gravitational fields. Unsuccessful policies do not necessarily correspond to valid solutions for balancing under different gravity forces. One important open problem is that of formally characterizing the family of parameterized RL problems in which sample reuse may be helpful.

CHAPTER 7
SUMMARY AND CONCLUSIONS

The main goal of this thesis is to improve the state of the art in autonomous acquisition of reusable skills. Our research is mainly driven by the desire to provide foundational methods assisting in the construction of open-ended learning agents capable of mastering their environments via learned libraries of composable skills.

One of the key reasons why humans—perhaps more than any other animals—manage to achieve competence in a wide variety of problems is their ability to acquire and refine skills. This idea is corroborated by recent research in cognitive and motor neuroscience, which shows that humans actively acquire general skills by identifying structural invariants in their environments. These structures can, once learned, be exploited to facilitate generalizing past learning experiences to novel problems.

In this thesis we proposed a computational framework to partially emulate the above-mentioned process. We introduced methods capable of identifying the lower-dimensional structures underlying the space of solutions to related problems. We exploit these lower-dimensional patterns to construct an abstract representation of an agent's behavior as it solves different problems. This is achieved by modeling how changes to a particular decision-making problem are reflected in the corresponding behaviors or policies adopted by the agent. Predictive models of this kind allow an agent to construct *parameterized skills*: procedures capable of producing appropriate solutions to novel problems, even those with which the agent has had no direct experience.

Parameterized skills are useful for agents that repeatedly encounter, throughout their lives, variations of a same overall decision-making problem. Instead of repeatedly incurring the cost of solving these problems from scratch, the agent might choose to learn a single reusable skill for producing appropriate behaviors to novel problems. This capability, combined with the possibility of sequentially and hierarchically combining learned skills, allows agents to produce increasingly more abstract and general behaviors.

Single parameterized skills may be internally composed of a series of specialized models of behavior, each one appropriate for solving a given subclass of problems. We showed that it is possible to autonomously identify these specialized behaviors by analyzing the topological properties of the policy manifold. In particular, we showed that specialized behaviors, which we refer to as sub-skills, are typically associated with particular lower-dimensional charts embedded in the policy manifold. We introduced a method to identify and model sub-skills, thereby allowing agents to hierarchically aggregate related parameterized behaviors into single, more abstract skills.

Finally, we observe that it may be advantageous for skill-learning agents to deliberately construct training regimens that allow them to more rapidly become competent. Skill acquisition in animals is often achieved by engaging in exploration, play and other curiosity-driven behaviors. Motivated by this observation, we developed an active learning framework capable of identifying training behaviors that lead to maximal skill competence progress.

We have empirically demonstrated that the methods introduced in this thesis allow for both simulated and physical robots to autonomously and efficiently acquire challenging parameterized skills. Nonetheless, much still remains to be done before truly open-ended learning robots can be constructed. The possible extensions detailed in Section 6 indicate that this work has only begun to scratch the surface of what is necessary to build such robots. Ultimately, it is our hope that the ideas introduced here represent a tangible, albeit possibly small step towards the practical construction of more broadly competent robots.

APPENDIX A
SMOOTHLY PARAMETERIZED CHARTS

One of the main assumptions made in this thesis is that smooth changes to the parameters τ of a problem result in smooth changes to its policy. If that is the case, it is possible to smoothly move over the manifold of policies by smoothly varying task parameters. This assumption is reasonable in a variety of situations, especially in the common case where a policy π_θ is differentiable with respect to its parameters.

As a motivating example, let us discuss a particular type of policy change that can induce a family of MDPs with smoothly varying goals. Consider perturbations $\Delta\theta$ to the parameters of a policy that do not change its path distribution p_θ by much:

$$\{\Delta\theta \in \mathbb{R}^M \text{ s.t. } \mathrm{KL}\big(p_\theta \| p_{\theta+\Delta\theta}\big) = \epsilon\}, \tag{A.1}$$

where p_θ $\mathrm{KL}(A\|B)$ is the KL-divergence between distributions A and B and ϵ is an arbitrarily small constant. If π_θ is differentiable, the second order Taylor expansion of $\mathrm{KL}(p_\theta \| p_{\theta+\Delta\theta})$ is $(\Delta\theta)^\top F(\theta)\Delta\theta$, where $F(\theta)$ is Fisher Information matrix:

$$F(\theta) = \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \pi_\theta(s,a)\nabla_\theta \log \pi(s,a) \nabla_\theta \log \pi(s,a)^\top \tag{A.2}$$

and $d^{\pi_\theta}(\cdot)$ is the stationary state distribution of the Markov chain induced by π_θ.

Consider, furthermore, a family of continuous-state MDPs that differ only in that they have different *goal states*—i.e., states that the agent tries to reach as rapidly as possible. Assume that π_θ is a policy that leads the agent, with high probability, to a goal state g. If we perturb the parameters of π_θ by $\Delta\theta$, such that $\Delta\theta$ respects the constraints in Equation A.1, the distribution of paths induced by π_θ and by $\pi_{\theta+\Delta\theta}$

remains similar. As a consequence, the perturbed trajectories induced by $\pi_{\theta+\Delta\theta}$ lead to goal states near g. Smooth changes to policy parameters, therefore, induce a set of smoothly varying goal states. An alternative interpretation is also possible: a set of smoothly varying goal states can be achieved by smoothly varying policy parameters. In this thesis we assume that this holds in more general settings: specifically, that smooth changes to the parameters τ of a problem result in smooth changes to its policy.

APPENDIX B
STANDARD ACQUISITION FUNCTIONS

Gaussian Processes are often used in Bayesian optimization to find the maximum of unknown, expensive-to-sample functions. To do so, surrogate *acquisition functions* are maximized (see Section 5.3). Acquisition functions guide the search for the optimum by identifying the most promising points to sample, so that the maximum can be rapidly identified. In Chapter 5 we introduced a novel acquisition function for identifying *tasks that maximize expected skill performance improvement.* Developing this new criterion was necessary because standard acquisition functions are not designed to select tasks which result in faster skill acquisition: if applied to a skill performance function, standard acquisition criteria identify tasks on which the agent is expected to achieve a high level of performance. Our goal, by contrast, is to identify training tasks that result in the highest *expected improvement in skill performance.*

In Section 5.3 we presented two standard acquisition function; we now discuss them in more details. Let $P(J(\tau)|\tau)$ be the current distribution over skill performance (as defined in Section 5.3) and $\mu(\tau)$ and $\sigma(\tau)$ be its mean and standard deviation, respectively. Furthermore, let P be the task distribution. The *Lower Confidence Bound* (LCB) acquisition criterion is defined as:

$$\text{LCB}(\tau) = \mu(\tau) - \kappa\sigma(\tau)$$

where κ is a tunable parameter. Intuitively, training tasks that minimize the LCB criterion causes the agent to focus on regions of the task space where skill performance is low but has a high probability of improvement. This criterion focuses on improving the agent's weaknesses, thereby trying to ensure that the skill will be minimally effective in the entire the range of possible tasks. This often leads agents to obsessively practice tasks that they may be unable to execute well.

Another commonly used acquisition criterion is Expected Improvement (EI). EI defines the expected performance improvement of a task τ over the task τ^+ with the highest performance observed so far as:

$$\mathrm{EI}(\tau) = \begin{cases} (\mu(\tau) - J(\tau^+) - \xi)\Phi(Z) + \sigma(\tau)\phi(Z) & \text{if } \sigma(\tau) > 0 \\ 0 & \text{if } \sigma(\tau) = 0, \end{cases}$$

where

$$Z = \frac{\mu(\tau) - J(\tau^+) - \xi}{\sigma(\tau)},$$

$\Phi(\cdot)$ and $\phi(\cdot)$ are the CDF and PDF of the Normal distribution, respectively, τ^+ is the task with the highest performance observed so far (which we denote, here, by $J(\tau^+)$) and ξ balances exploration and exploitation. If applied to the problem of selecting training tasks to efficiently learn a parameterized skill, EI would select tasks whose individual performances are expected to improve the most by practice. This may lead the agent to repeatedly practice tasks that are already well-modeled by the skill but which may be marginally improved; it also causes the agent to ignore regions of the task space in which it is not yet competent.

APPENDIX C
THE CATAPULT DOMAIN

We simulated launches in the Catapult Domain (Section 5.6) using standard ballistic equations. We assumed Earth's gravity.

Assume that a given projectile is launched from the catapult with velocity v and angle θ, and let C_x and C_y be, respectively, the horizontal and vertical coordinates of the catapult. We can decompose the velocity vector of the projectile in its horizontal and vertical components: $v_x = vcos(\theta)$ and $v_y = vsin(\theta)$. The projectile follows a parabolic path $y(t) = ax(t)^2 + bx(t) + c$, where $x(t)$ and $y(t)$ are the horizontal and vertical coordinates of the projectile at time t, respectively, and a, b and c are parabola parameters:

$$\begin{aligned}
a &= \frac{-9.8}{2v^2cos(\theta)^2} \\
b &= \frac{v_y}{v_x} + \frac{9.8C_x}{v_x^2} \\
c &= C_y - \frac{v_yC_x}{v_x} - \frac{1}{2}\frac{9.8C_x^2}{v_x^2}
\end{aligned}$$

The point of impact of the projectile is given by the point where its parabolic path intersects the equation describing the terrain. If using a piecewise-linear approximation of the terrain surface, the impact point can be found by identifying the coordinates where the parabolic path of the projectile first intersects one of the line segments describing the terrain.

APPENDIX D

EISP UNDER A SQUARED EXPONENTIAL KERNEL

Analytical solutions to Equations 5.5 and 5.6 depend on the choice of kernel. If we assume that P is a uniform distribution over tasks and define k_S as the squared exponential kernel $k_S(\tau_1, \tau_2) = \sigma_f^2 \exp(-L^{-1}(\tau_1 - \tau_2)^2)$, then:

$$\begin{aligned}
\nabla_\tau k_C(\tau^t, \tau_i^{t_i}) &= -\frac{2}{L}(\tau - \tau_i) k_C(\tau^t, \tau_i^{t_i}) \\
g_t(\tau_i^{t_i}) &= \frac{1}{2}\left(\sigma_f^2 \sqrt{\pi L}\right) k_T(t+1, t_i) \times \\
&\quad \left[\operatorname{erf}\left(\frac{\tau_i - A}{\sqrt{L}}\right) - \operatorname{erf}\left(\frac{\tau_i - B}{\sqrt{L}}\right)\right] \\
\nabla_\tau g_t(\tau_i^{t_i}) &= \sigma_f^2 k_T(t+1, t_i) \times \\
&\quad \left[\exp\left(\frac{-(A - \tau_i)^2}{L}\right) - \right. \\
&\quad \left. \exp\left(\frac{-(B - \tau_i)^2}{L}\right)\right].
\end{aligned}$$

where erf(z) is the Gauss error function.

APPENDIX E
A METHOD FOR EFFICIENTLY PRACTICING TASKS

Practicing sample tasks to construct the training set K is the most time-intensive step involved in acquiring a parameterized skill. Several policy learning algorithms can be used to identify an optimal or near-optimal policy for a given task. Most algorithms, however, explicitly balance exploration and exploitation while doing so. This is necessary whenever the agent has to learn a good actuation policy while *at the same time* obtaining as much reward as possible.

Often, however, it makes sense to assume an initial training phase during which the goal of the agent is to just explore efficiently, so that an optimal or near-optimal policy can be learned fast but without necessarily worrying about performing well (see, e.g., the work of Şimşek and Barto [15]). This is precisely the problem encountered by an agent practicing tasks to construct a skill: it is only concerned with rapidly identifying a near-optimal policy for each training task, but does not care about performing well while doing so.

In this chapter we study the problem of finding exploration policies for the case in which an agent is momentarily not concerned with exploiting, and instead tries to compute a near-optimal policy for later use. We formally define the Optimal Exploration Problem as one of sequential sampling by posing it as an MDP constructed by expanding the state space of the original one that we want to explore. Solutions to this expanded MDP correspond to paths of minimum expected length in the space of policies and describe optimal sequential sampling trajectories. We show an important property of such solutions and a special function that can be constructed based on them. Since directly computing these solutions is not feasible, we derive a local linear approximation to the relevant estimates and present an intuitive geometric interpretation of its meaning. We compare our model-free approach to other exploration techniques, most importantly Delayed Q-Learning (DQL) [68] and ΔV [15], and show that ours is both based on a well-defined optimization problem and empirically efficient.

Note that the method introduced in this chapter is only applicable to MDPs with discrete state and action spaces. This simplifies the problem but somewhat limits the applicability of the resulting algorithm to continuous parameterized skill acquisition settings. In the experiments described in Sections 3.5, 4.4 and 5.6, therefore, we carefully selected alternative policy learning algorithms that were known to perform well at those particular tasks.

E.1 Summary of the Method

We first informally describe our method and in later sections show that it constitutes a principled approximation to a well-defined optimization problem. The algorithm that we propose, which we call $\Delta\pi$, focuses exploration not on regions where the value function is changing the most, or in which a model is being made more accurate, as several of the works described in Section E.5, but on regions where the *likelihood of a change in the policy is high.* In other words, exploration is based on how sensitive the policy is at a given state, given that we continue to gather information about it.

The indicator of policy sensitivity that we use is based on a simple linear extrapolation of the behavior of the Q-function at a state, both for the action currently considered to be optimal and for some other recently tried action. Specifically, given any unbiased estimate of how a Q-value changes as more samples are collected (e.g., a temporal difference error), we can estimate if and *when* the value of some action with a lower Q-value will surpass the value of the one currently considered to be optimal. When exploring, one should find desirable those actions whose Q-values are soon likely to surpass that of the action currently considered to be optimal; the sooner this crossing point is predicted to occur, the more attractive the action should be. If, on the other hand, the evolution of Q-values indicates that the ordering of actions is likely to remain the same, then we shouldn't find those states attractive (Figure E.1). We denote an approximation to the *expected time until a policy change* in a given state by $d(s, a)$. This value serves as a guide to how valuable it is to explore certain parts of the MDP based on how likely it is that new samples from them will lead to changes in the policy. The derivation of $d(s, a)$ as a principled approximation and its precise definition are given in Sections E.2 and E.3.

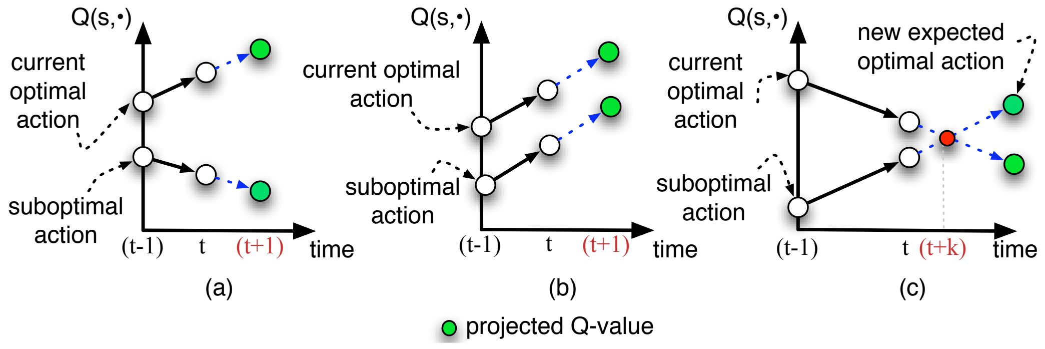

Figure E.1. Geometric interpretation of the expected time until a policy change, $d(s, a)$, represented by the intersection between lines. (a) Q-values are predicted to diverge and no change is expected; (b) Q-values seem to evolve at same rate and no change is expected; (c) a change expected in k steps.

Notice that negative expected times until a policy change have a natural interpretation: a change has *already* happened and now the Q-values seem to be diverging (Figure E.1a). Our exploration algorithm considers small values of $|d(s, a)|$ attractive, since they either indicate that a policy change is expected soon or that one has happened recently. In the latter case, it might be important to continue exploring the corresponding states to ensure that the change was not caused by noise in the sampling of rewards and next states. In order to model this, we do not use $d(s, a)$ directly as an indication of policy sensitivity; instead, we define the following quantity:

$$r(s,a) = \begin{cases} \exp\left(-\frac{d^2(s,a)}{\sigma}\right) & |d(s,a)| < \lambda \\ -p & |d(s,a)| \geq \lambda, \end{cases} \tag{E.1}$$

where p is a small penalty given when the action's values seem to have stabilized, λ quantifies how rigorous we are when deciding whether this is the case, and σ controls the maximum horizon of time during which we trust the predictions made by our local linear approximation. In practice we have observed that many functions other than the Gaussian can be used to define $r(s, a)$, as long as they are monotonically increasing and decreasing in the same intervals as a Gaussian and the resulting $r(\cdot, \cdot)$ is bounded. In systems where noise does not play a crucial role, one might want to favor exploration of expected *future* policy changes by adding a penalty (e.g, -1) to $r(s, a)$ in case $d(s, a) < 0$.

We point out that the direct use of $r(s, a)$ as a guide for exploration provides just a myopic perspective, since it reflects only the value of exploring *one specific state and action*. In general, though, the choice of which regions to explore is a sequential decision problem: states that do not look promising now might allow the agent to reach regions where several corrections in the policy are expected. This can be dealt with by using $r(s, a)$ as a *new*, surrogate reward function for the MDP that we want to explore, in which case its solutions approximately minimize the sum of times until all policy corrections are performed. This corresponds to executing an exploration policy that tries to correctly rank actions as fast as possible (see Section E.2). Notice that because $r(s, a)$ is used as a surrogate reward function, we need to store the original Q-function (the one related to the exploitation policy being estimated) separately. Specifically, we keep track of two separate Q-functions: one related to the *exploration policy* and one to the *exploitation policy*. The latter is constructed based on samples collected by the former, and the former is updated given new estimates from the latter. This is also the approach taken by Şimşek and Barto [15].

E.2 Optimal Exploration

In this chapter we use a slightly different notation than the one presented in Section 2.1. This will allow for a simpler description of the analyses and algorithms introduced in the following sections.

Let an MDP M be a tuple $\big(S_M, A_M, R_M, T_M, \gamma_M\big)$, where S_M is a finite set of states, A_M is a finite set of actions, $R_M : S_M \to \mathbb{R}$ is a reward function, $T_M : S_M \times A_M \times S_M \to [0, 1]$ is a transition function, and γ_M is a discount factor. Solving M consists of finding an optimal policy π^*_M, i.e., a mapping from states to actions that maximizes the expected discounted sum of future rewards. Let $Q^\pi_M(s, a)$ be the function that gives the expected total discounted reward obtained when taking action a in s and following π thereafter. The optimal Q-function for an MDP M is denoted by Q^*_M, and an estimate of it at time t by Q^t_M. A greedy policy with respect to a Q-function can be derived by taking the action that maximizes the Q-function at a given state; let $\pi_{[Q]}$ be this deterministic greedy policy, obtained when using Q to rank actions and breaking ties randomly. Let $V^\pi(s)$ be the value of state s when following policy π, and $V_D(\pi)$ be the value of a policy π given an initial state distribution: $V_D(\pi) = \sum_{s\in S} D(s)V^\pi(s)$, where $D(s)$ is the probability of the MDP starting at state s.

For the problem of *Optimal Exploration*, we wish to find a (possibly non-stationary) policy such that the samples it collects allow for the identification of π^*_M as quickly as possible; we note that this is different from calculating V^*_M as quickly as possible. Specifically, an optimal exploration policy might correctly rank all *optimal* actions even though the values of some (or all) states are still inaccurate. Formally, we define the Optimal Exploration Problem as one of **sequential sampling** by posing it as an

MDP constructed by expanding the state space of the process we originally want to explore. Solutions to this expanded MDP correspond to paths of minimum expected length in the space of policies and describe optimal sequential sampling trajectories. Based on the original MDP M, we define a new MDP, M', such that any optimal policy for M', by construction, induces an optimal exploration strategy for M. As will become clear shortly, optimality is defined in terms of the minimum expected number of actions (or steps) needed until enough information is collected and π^*_M can be found. We construct M' in a way so that trajectories in it correspond to sequences of joint evolution of states in M and estimates Q^t_M; this evolution satisfies the Markov property and encodes trajectories in the space of policies for M. M' is defined by:

- a state space $S' = S_M \times \mathbb{R}^{|S_M||A_M|}$. S' corresponds to the same state space of M, but augmented with the current estimate of the optimal Q-function for M. We denote the state $s'_t \in S'$ in which M' is at time t as a tuple $s'_t \equiv \left(s^t_M, Q^t_M\right)$;

- an action space $A' = A_M$, i.e., the same as the action space of the original MDP;

- Q^0_M, an initial estimate of Q^*_M;

- L, an off-policy, deterministic learning mechanism that converges to an optimal policy. Given an action a taken in M', we can imagine also executing a in the original MDP, M, and observing a sample experience $\left(s^t_M, a, r^t_M, s^{t+1}_M\right)$. L then takes this information, along with Q^t_M, and returns an updated estimate of the Q-function: $Q^{t+1}_M \leftarrow L(s^t_M, a, r^t_M, s^{t+1}_M, Q^t_M, \rho)$, where ρ is the set of any other data structures or parameters required by L, such as step size parameters, models, etc;

- T', a transition function based on T_M and L. Given the current state s'_t of M', T' describes the distribution over possible next states s'_{t+1}. Since s'_{t+1} is a tuple of the form $\left(s^{t+1}_M, Q^{t+1}_M\right)$, we can think of T' as computing each of those components independently: s_M^{t+1} probabilistically according to $T_M(s_M^t, a)$, and Q_M^{t+1} by applying L to the last sample experience obtained when executing a in M. Also, T' is such that all states with zero instantaneous reward (i.e., goal states, as defined below) are absorbing;

- $0 < \gamma' < 1$, a (fixed) discount rate;
- R', a reward function mapping states of M' to the reals:

$$R'((s_M^t, Q_M^t)) = \begin{cases} -1 & \text{if } V_D(\pi_{[Q_M^t]}) \neq V_D(\pi_M^*) \\ 0 & \text{otherwise.} \end{cases}$$

Note that rewards in M' are nonnegative only in states in which the use of the best actions, according to the ranking induced by Q_M^t, yields a greedy exploitation policy for M whose value is optimal. This ensures that maximizing cumulative rewards in M' is equivalent to efficiently reaching a Q-function for M that allows all *optimal* actions to be correctly ranked. This is made rigorous in Proposition 1:

Proposition 1 *An optimal policy for M' specifies a path of minimum expected length in the space of policies for M, starting from an arbitrary initial policy and reaching an optimal policy for M. Paths between policies are specified by sequences of sample experiences in M.*

Proposition 1 follows from the facts that *(1)* S_M and A_M are finite and L is deterministic, and thus from any $s' \in S'$ there exists only a finite number of possible next states in M'; *(2)* since R' is bounded and $0 < \gamma' < 1$, the value function for M' is bounded, specifically in $[\frac{1}{\log \gamma'}, 0]$; and finally *(3)* because L is a learning algorithm that converges to an optimal policy for M (even if asymptotically), there exists *at least one* proper policy for M', that is, one that reaches the goal state with probability 1 regardless of the initial state. This is true because otherwise M would not be solvable. Taken together, these observations imply that there exists a nonempty, possibly uncountable number of proper policies for M', which form a totally ordered set with respect to the value of each policy. Because this set is bounded above, its supremum is well-defined and there exists an optimal policy for M'. This policy, by construction, minimizes the expected number of samples needed in order to compute π^*_M. All above-mentioned expectations are taken over all possible trajectories in M'. This result is similar to the more general problem of Stochastic Shortest Paths (SSP) [8] — the main difference being that SSPs require MDPs with finite state spaces. Finally, note that M' is constructed in such a way that an optimal policy for M is reached whenever the greedy policy induced by the current estimate Q^t_M correctly ranks all optimal actions, *even if the values of the states themselves are still inaccurate.*

It should be clear that directly solving M' is not feasible, since R' assumes prior knowledge of an optimal policy for M. This impossibility is not surprising: one cannot find a *truly* minimal sequence of exploration actions without knowing beforehand T_M and R_M, which would make exploration unnecessary. However, M' is useful since we can observe general properties of its solutions and use them to construct a principled technique for efficient exploration. In what follows, we discuss some of these properties and derive a local linear approximation which allows us to construct a principled exploration strategy called $\Delta\pi$.

Let $\phi^{\pi}_{s,a}(t)$ be the expected value of $Q_M(s,a)$ after a trajectory of length t in M', starting from some given state $s' \in S'$ and following a fixed policy π for M'. In order to simplify the notation, we suppress the dependence on s':

$$\phi^{\pi}_{s,a}(t) = E[Q^t_M(s,a)]. \tag{E.2}$$

The above expectation is taken with respect to trajectories in M'; the probabilities involved depend on π and T'. ϕ encodes how Q-value estimates are expected to evolve if updated with samples collected by π. Let π_{expl} be any policy for M'; this policy induces an exploration policy for M. Let us analyze the expected length k of the shortest trajectory in M', when following π_{expl}, such that we expect a change in the greedy policy (for M) induced by the expected Q-values: $\arg\min_k \left[\exists s \in S \quad \pi_{[\phi^{\pi_{expl}}(t+k)]}(s) \neq \pi_{[\phi^{\pi_{expl}}(t)]}(s)\right]$. If this is generated by an optimal policy for M', then k is the expected minimum number of samples from M needed to cause a change in the current greedy policy. Similarly, we can define the expected minimum number of samples until the induced policy changes *in a given state* $s \in S$:

$$\arg\min_k \left[\pi_{[\phi^{\pi_{expl}}(t+k)]}(s) \neq \pi_{[\phi^{\pi_{expl}}(t)]}(s)\right]. \tag{E.3}$$

Let us now assume we have taken an arbitrary step in M' and observed a next state $s'_{t+1} \in S'$. This state contains an updated estimate of the Q-function, namely Q_M^{t+1}. If the ranking of actions induced by Q_M^t changes with respect to Q_M^{t+1}, we say a *crossing* has occurred. For example, if a_1 and a_2 are actions and $Q_M^t(s, a_1) > Q_M^t(s, a_2)$ but $Q_M^{t+1}(s, a_1) \leq Q_M^{t+1}(s, a_2)$, then a crossing has occurred.

Note that ϕ is defined only in the domain of integer timesteps. For our purposes, however, it is advantageous to embed it in a continuous process by assuming that updated Q-values change *linearly and continuously* between timesteps. Viewing ϕ as a function of continuous time is useful for the following reason: if $Q_M^t(s, a_1) > Q_M^t(s, a_2)$ but $Q_M^*(s, a_1) \leq Q_M^*(s, a_2)$, then for some (not necessarily integer) k, $\phi^\pi_{(s,a_1)}(t+k) = \phi^\pi_{(s,a_2)}(t+k)$, assuming that π is a proper policy for M'. This proposition is trivially true because of the Intermediate Value Theorem. It allows us to say that a crossing occurs precisely at the time k when the Q-values of two actions are momentarily equal, before one surpasses the other. It also helps us to interpret non-integer values of ϕ, which might occur since it is an expectation. Finally, it makes it easier to meaningfully compare non-integer expected crossing times in terms of the *rate* with which the ranking of actions seems to be changing. This becomes particularly clear if using Boltzmann policies with high temperatures, in which case the rate of change in action probabilities of two actions, as new samples are collected, can be shown to cross exactly when the derivatives of their Q-values becomes equal. This connection between the rate of change in action preferences and the derivative of their Q-values appears again as part of the solution of Equation E.5.

E.3 Deriving an Efficient Exploration Policy

We would now like to use the definition of ϕ (or an approximation of it) to derive an efficient, though not necessarily optimal, exploration strategy for M. We first observe that because updates to the Q-function are generally not independent, the *minimum time to rank actions in all states* (the quantity minimized by $\pi^*_{M'}$) is not equal to the sum of the minimum times to rank *actions at each state* in turn. However, we propose that a policy that minimizes the latter is also a good approximation of the former. We further note that because $d(s, a)$ is an estimate of the minimum time until a change in ranking *at a given state*, it is possible to minimize that latter quantity by solving a sequential decision process in which $d(s, a)$ (or a related quantity) serves as a surrogate reward function for M. Under this new reward structure, π^*_M defines an *efficient exploration policy* which quickly improves the ranking of actions at each state. For more details, see Algorithm 1. We empirically show this to be an effective approximation in Section E.4 and discuss when it might perform poorly in Section E.6. Finally, note that minimizing $d(s, a)$ is equivalent to *minimizing the time until the nearest crossing*. Let us build on this last observation and define $c_{(s,a_1,a_2)}(t)$, the expected difference between the Q-values of any actions a_1 and a_2, for any given state $s \in S$:

$$c_{(s,a_1,a_2)}(t) = \phi^{\pi}_{(s,a_1)}(t) - \phi^{\pi}_{(s,a_2)}(t). \tag{E.4}$$

The smallest root of $c_{(s,a_1,a_2)}(t)$ corresponds to the *minimum expected time* at which a_1 and a_2 cross, and therefore represents exactly the information required for estimating $d(s, a)$. However, ϕ^{π} (and therefore c as well) is hard to describe analytically since the precise understanding of how Q-values evolve requires knowing the structure of M and of the learning algorithm. Although we do not have a closed form for ϕ^{π}, we can use a Taylor expansion around the time of the *last* sample experience, $t-1$:

$$\hat{\phi}^{\pi}_{(s,a)}(t) \approx \phi^{\pi}_{(s,a)}(t-1) + \frac{\partial \phi^{\pi}_{(s,a)}(t-1)}{\partial t}. \tag{E.5}$$

We expand the series around the time of the last experience since we need to approximate the terms in Equation E.5 by using sampled values; it should be clear that any statistics of interest will be the most accurate if we allow the use of all $t-1$ samples observed so far. Also, note that we *do* have unbiased samples for both terms in Equation E.5: a sample of $\phi^{\pi}_{(s,a)}(t-1)$ is simply $Q^{t-1}_{\pi}(s,a)$, and a sample of $\frac{\partial \phi^{\pi}_{(s,a)}(t-1)}{\partial t}$ is $\alpha_M \delta_{(s,a)}(l)$, where $\delta_{(s,a)}(l)$ is the TD error[1] for the last time $Q_{\pi}(s,a)$ was updated, at time l; α_M is the step-size parameter used in L. For any given π, these are unbiased estimators: $Q^{t-1}_{\pi}(s,a)$, directly because of the definition of $\phi^{\pi}_{(s,a)}(t-1)$; and $\alpha_M \delta_{(s,a)}(l)$, by a similar argument and by noticing that *(1)* it can computed by subtracting consecutive Q-values, and *(2)* expectation is a linear operator. Better, lower-variance estimates of the derivative of $\phi^{\pi}_{(s,a)}(t)$ can be obtained and are useful in highly stochastic problems: one could estimate them via finite differences, by averaging past updates to the Q-function, or by propagating updates to other Q-values through a model. In what follows, we use just the simplest estimates possible, as described above, and instantiate a model-free version of $\Delta\pi$ called TD(0)-$\Delta\pi$.

Proposition 2 *A local linear approximation to $\phi^{\pi}(s,\cdot)(t)$ induces a family of approximations for the functions $c_{(s,\cdot,\cdot)}(t)$, whose* smallest roots *correspond to approximations of the minimum expected time until a crossing between any pair of actions.*

[1]TD errors are not required, though; any observed difference between consecutive estimates of a Q-value suffice.

Proposition 2 follows from simple geometric reasoning based on $\hat{\phi}^{\pi}$ being a linear approximation. Specifically, we can show that a local linear approximation to the expected time until the value of an action a crosses the value of the one currently considered optimal, $\hat{a}^*$, for some $s \in S$, is:

$$\begin{aligned} d(s,a) &= \left(\frac{1}{\alpha_M}\right) \frac{Q^t(s,\hat{a}^*) - Q^t(s,a)}{\delta_{(s,a)}(T_{s,a}) - \delta_{(s,\hat{a}^*)}(T_{s,\hat{a}^*})} \\ &\approx \arg\min_t \big(c_{(s,a,\hat{a}^*)}(t) = 0\big) \end{aligned} \tag{E.6}$$

where T_{s,a_i} is the last time at which $Q(s,a_i)$ was updated. $d(s,a)$ is a valid approximation unless its denominator is zero, which occurs if both Q-values seem to be changing at the same rate — in this case, it correctly concludes that no crossings are expected. Note also how it implements the type of policy sensitivity indicator described in Section E.1.

E.4 Experiments

We now compare our approach to other algorithms for efficient exploration. Our main comparisons are with ΔV [15] and Delayed Q-Learning (DQL) [68]. ΔV is a principled, model-free way of finding efficient, purely-exploratory policies. DQL is, to the best of our knowledge, the model-free technique with best PAC-MDP bounds, and provably performs near-optimally in all but a polynomial number of timesteps. We also compare with two baseline algorithms: *(1)* a Constant-Penalty (CP) technique, which gives small penalties to each visited state and thus implements a *least-visited* strategy [80]; and *(2)* ϵ-greedy Q-Learning (QL), for several values of ϵ; this includes random exploration ($\epsilon = 1$).

The first domain in which we evaluate TD(0)-$\Delta\pi$ consists of a simple discrete 25×25 maze with four exits. The four usual actions are available (N,S,E,W), and

Algorithm 1 TD(0)-$\Delta\pi$

```
for all $(s, a)$ do
    $Q^0_{exploit}(s, a) \leftarrow 0$;   $Q^0_{explore}(s, a) \leftarrow 0$;
    $\delta_{(s,a)}(0) \leftarrow 0$;   $T_{s,a} \leftarrow 0$;   visited$(s, a) \leftarrow False$
end for
for $t = 1, 2, 3, \ldots,$ do
    Let $s_t$ be state of $M$ at time $t$
    Choose action $a_t := \arg\max_{a' \in A_M} Q^t_{explore}(s, a')$
    Take $a_t$ in $M$, observe reward $r^t_M$, next state $s'$
    Let $\hat{a}^* := \arg\max_{a' \in A_M} Q^t_{exploit}(s, a')$
    if not visited$(s_t, a_t)$ or not visited$(s_t, \hat{a}^*)$ then
        $r(s_t, a_t) := 1$
    else
        if $|\delta_{(s_t,a_t)}(T_{s_t,a_t}) - \delta_{(s_t,\hat{a}^*)}(T_{s_t,\hat{a}^*})| < \lambda$ then
            $r(s_t, a_t) := -p$
        else
            Compute $r(s_t, a_t)$ according to Eq. E.1 and E.6
        end if
    end if
    $Q^{t+1}_{exploit} \leftarrow L(s^t, a_t, r^t_M, s', Q^t_{exploit}, \rho_{\mathbf{exploit}})$
    $Q^{t+1}_{explore} \leftarrow L(s^t, a_t, r(s_t, a_t), s', Q^t_{explore}, \rho_{\mathbf{explore}})$
    $T_{s_t,a_t} \leftarrow t$;   visited$(s_t, a_t) \leftarrow True$;
    $\delta_{s_t,a_t}(t) \leftarrow Q^{t+1}(s_t, a_t) - Q^t(s_t, a_t)$;
end for
```

each has a 0.9 probability of taking the agent to the intended direction, and 0.1 of taking it to another uniform random direction. Rewards are -0.001 for each action, and 1, 2 or 5 when transitioning into one of the terminal states. Q-functions in Algorithm 1 are learned using Q-Learning with step-size parameter $\alpha = 0.1$ and discount rate $\gamma = 0.99$.

Results for the value of the learned exploitation policy as a function of the amount of exploration allowed are shown in Figure E.2 and E.3, and are averages over 20 runs. Both our approach, ΔV and DQL perform significantly better than the baseline algorithms. We searched the space of values of ϵ, for QL, and present only some sample results. ΔV initially performs better than our approach, mainly because the random walk it performs during its initial phase finds one of the goals faster; however, a closer look reveals that it gets "obsessed" with fine-tuning the value of states even when the policy for reaching them is already correct. At this moment, on the other hand, TD(0)-$\Delta\pi$ notices that no other policy changes are expected and proceeds to other regions of the state space. TD(0)-$\Delta\pi$ finds the optimal exploitation policy almost 100,000 steps before ΔV. DQL takes even longer to learn: in principle it requires (for this domain) $m \approx$*1 billion* samples before updating the value of any given state–action pair, in order for its bounds to guarantee convergence in polynomial time. In our experiments we used more reasonable values for m, which removed its PAC-MDP properties but made it comparable to other approaches. DQL's bounds also require Q-values to be initialized optimistically, which for this domain means setting $Q_0(s, a) = 500$. However, we noticed that only values of $Q_0(s, a) \leq 9$ were capable of generating reasonable learning curves. Furthermore, the only way we could make DQL perform similarly to TD(0)-$\Delta\pi$ was to initialize its Q-values fairly close to the *optimal* ones, and even then it became stuck in a local minimum 20% less efficient than the optimal exploitation policy. We searched the space of parameters of DQL to make it perform as well as possible; a representative sample of the learning curves is shown in Figure E.3.

The second domain in which we evaluate TD(0)-$\Delta\pi$ is a rod positioning task [49], which consists of a discretized space containing a rod, obstacles, and a target. The goal is to maneuver the rod by moving its base and angle of orientation so that its tip touches the target, while avoiding obstacles. We discretize the state space into unit x and y coordinates and $10°$ angle increments; actions move the rod's base one unit in either direction along its axis or perform a $10°$ rotation in either direction. Rewards are -1 for each action and 1000 when the tip of the rod touches the goal. We used the same learning method and parameters as in the previous domain. Results for the value of the learned exploitation policy as a function of the amount of exploration allowed are shown in Figures E.4 and E.5. $\Delta\pi$ again performed better than other methods; interestingly, simple approaches like ϵ-greedy QL and CP performed *better* than specialized ones such as ΔV and DQL — the reason being that this domain contains only *one* source of positive reward, which, when found, can be aggressively exploited without risking overlooking others. ΔV again kept fine-tuning the value function even when the policy was already correct, and often got stuck in local minima 25% worse than the optimal exploitation policy. $\Delta\pi$, on the other hand, explored a region only while it had evidence that the policy could still change. We searched the space of parameters of DQL to optimize its performance; representative learning curves are shown in Figure E.5. DQL only performs well if initialized with a Q-function fairly close to the optimal and if m is set much lower than required to guarantee its PAC-MDP bounds. After 1.8 million timesteps, it learned an exploitation policy 50% worse than the optimal one.

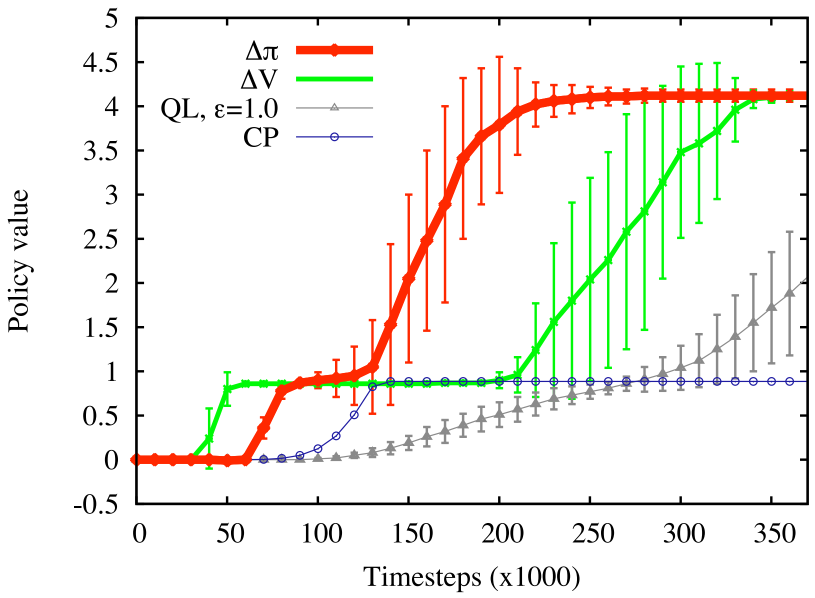

Figure E.2. Performance of $\Delta\pi$ in the maze domain.

E.5 Related Work

Efficient exploration in RL has been studied extensively, usually with the objective of maximizing return in an agent's lifetime, thus requiring a trade-off between explo-

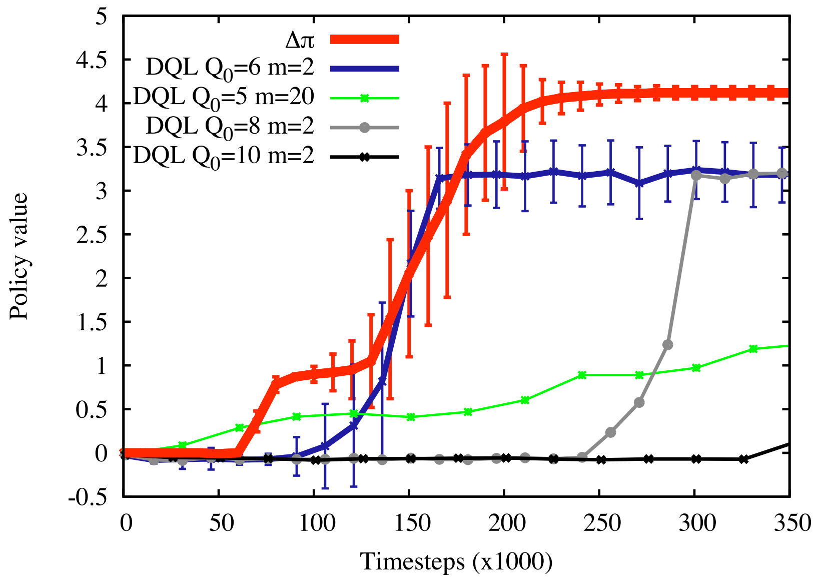

Figure E.3. Performance of $\Delta\pi$ in the maze domain (vs. DQL).

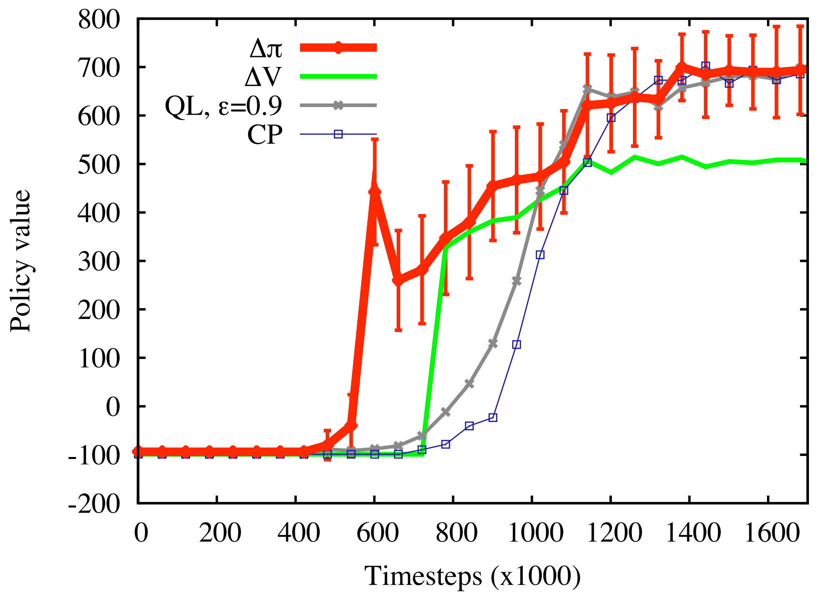

Figure E.4. Performance of $\Delta\pi$ in the rod positioning domain.

ration and exploitation. In this chapter, on the other hand, we are concerned with purely exploratory policies. Some of the existing approaches to tackle this are simple techniques such as random exploration, picking actions that were selected the least number of times, visiting unknown states, etc. [80]. These are inefficient due to treat-

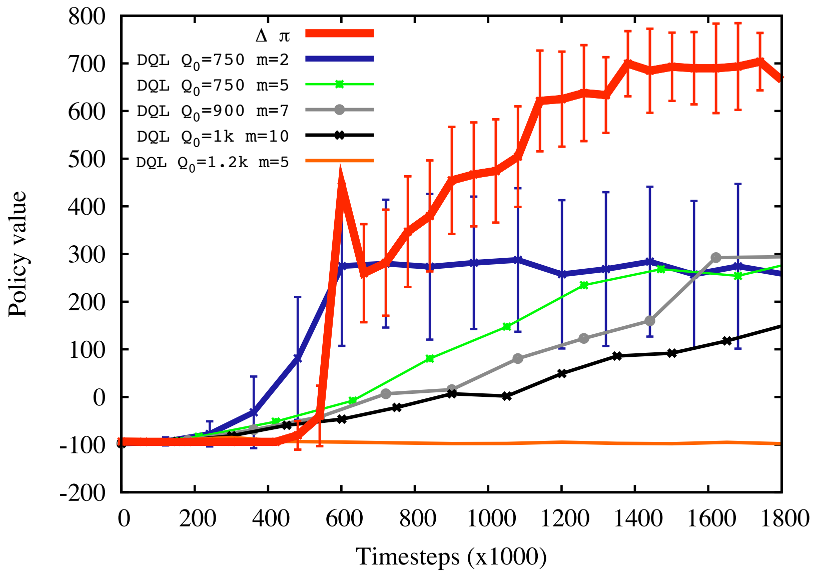

Figure E.5. Performance of $\Delta\pi$ in the rod domain (vs. DQL).

ing the entire state space uniformly, ignoring useful structure provided by the value function. Other approaches for efficiently learning consider the full exploration versus exploitation problem directly. Duff [21] proposes a Bayesian approach for the case in which prior uncertainties about the transition probabilities are available; Abbeel and Ng [1] present a method for computing a near-optimal policy assuming that demonstrations by a teacher are available. Another model-based Bayesian approach was proposed by Dearden, Friedman, and Andre [18], where the Value of Information for exploring states is computed considering a model and uncertainty about its parameters. Finally, Kolter and Ng [35] present a method for constructing a belief state for the transition probabilities and obtaining a greedy approximation to an optimal Bayesian policy.

Two techniques have been especially influential among researchers studying efficient RL algorithms: R–Max [11], and E^3 [30]. Both give polynomial guarantees for the time to compute a near-optimal policy. These techniques differ from ours in at least two important aspects: *(1)* they maintain a complete, though possibly inaccurate model of the environment; *(2)* they perform expensive, full computations of policies (via, e.g., value iteration) over the known model as steps in their algorithms. Therefore, a direct, meaningful comparison with our **model-free** approach would be difficult. Instead, we compare with Delayed Q-Learning [68], a model-free approach which, to the best of our knowledge, has the best known PAC-MDP bounds and which provably performs near-optimally in all but a polynomial number of timesteps.

Other techniques relevant to this work include the Active RL algorithm [22] and the ΔV approach [15]. The former is similar to ours in that it defines an exploration policy based on a type of sensitivity analysis, namely that of the policy with respect to perturbations to a model. It differs from ours in that our analysis focuses, alternatively, on the impact that collecting additional samples has on the expected evolution of Q-values, and therefore on the ranking of actions, and also in that Active RL assumes an initial, complete estimate of a model, while we don't. The latter approach (ΔV) shares with ours the idea of model-free exploration and uses a similar formalization. It focuses exploration on regions of the state space where the magnitude of the value function changes the most, implicitly maximizing the speed with which the value function is fine-tuned. Unfortunately, it has practical shortcomings, mainly because agents following it become "obsessed" with fine-tuning the value of states even when the policy is already correct. Intuitively, the specific values of the states shouldn't matter; the important information to be acquired is the *ranking* of actions. Achieving this type of exploration strategy is the goal of this chapter.

E.6 Discussion

We have presented a derivation of a local linear approximation to the expected time until a policy change and used it to construct an efficient, model-free exploration technique. The specific approximation used might have practical shortcomings. It is possible, for instance, to construct MDPs in which TD(0)-$\Delta\pi$ performs poorly by initializing it in a region of the state space where many crossings are likely to occur but which is not part of any optimal trajectory. We believe, however, that these cases are not common in practice. In fact, ΔV seems much more sensitive to small changes in the formulation of the MDP, since simply rescaling the reward function can make it perform arbitrarily slowly. DQL, even with provably polynomial sample complexity, is a good example of how such guarantees don't necessarily correspond to algorithms that are feasible in practice.

For future work, we would like to study model-based estimations of Equation E.5, which could have lower variance. We also believe there might be a relevant connection between Equation E.6 and Advantage functions, and that PAC-MDP bounds can be obtained. Another interesting open problem is that of deciding when to safely terminate the exploration process.

Even more important is the question of whether this method can be extended to the case of MDPs with continuous states and actions. Being able to identify efficient exploration policies under this alternative formulation would allow an agent to more rapidly identify near-optimal policies for various training tasks, thereby further accelerating the construction of the skill training set K.

BIBLIOGRAPHY

[1] Abbeel, P., and Ng, A. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning* (2005), pp. 1–8.

[2] Baranes, A., and Oudeyer, P. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems 61*, 1 (2013), 69–73.

[3] Barrett, S., Taylor, M., and Stone, P. Transfer learning for reinforcement learning on a physical robot. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop* (2010).

[4] Barto, A. Intrinsic motivation and reinforcement learning. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, G. Baldassarre and M. Mirolli, Eds. Springer, 2013, pp. 14–47.

[5] Barto, A. G., Singh, S., and Chentanez, N. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the Third International Conference on Developmental Learning* (2004), pp. 112–119.

[6] Baxter, J. Bartlett, P. L. Direct gradient-based reinforcement learning. In *Proceedings of the International Symposium on Circuits and Systems* (2000), vol. 3, pp. 271–274.

[7] Berthier, N., and Keen, R. Development of reaching in infancy. *Experimental Brain Research 169* (2006), 507–518.

[8] Bertsekas, D.P., and Tsitsiklis, J. N. An analysis of stochastic shortest path problems. *Mathematics of Operations Research 16*, 3 (1991), 580–595.

[9] Bhatnagar, S., Sutton, R., Ghavamzadeh, M., and Lee, M. Natural actor-critic algorithms. *Automatica 45*, 11 (2009), 2471–2482.

[10] Bitzer, S., Havoutis, I., and Vijayakumar, S. Synthesising novel movements through latent space modulation of scalable control policies. In *Proceedings of the Tenth International Conference on Simulation of Adaptive Behavior: From Animals to Animats* (2008), pp. 199–209.

[11] Brafman, R.I., and Tennenholtz, M. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research 3* (2001), 213–231.

[12] Braun, D., Waldert, S., Aertsen, A., Wolpert, D., and Mehring, C. Structure learning in a sensorimotor association task. *PLoS ONE 5* (2010), e8973.

[13] Brock, O., Fagg, A., Grupen, A.R., Platt, R., Rosenstein, M., and Sweeney, J. A framework for learning and control in intelligent humanoid robots. *International Journal of Humanoid Robotics 2(3)* (2005).

[14] Şimşek, O., and Barto, A.G. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-first international conference on Machine learning* (2004), pp. 751–758.

[15] Şimşek, Ö., and Barto, A.G. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd International Conference on Machine learning* (2006), pp. 833–840.

[16] Şimşek, O., Wolfe, A., and Barto, A.G. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second international conference on Machine learning* (2005), pp. 816–823.

[17] da Silva, B.C., Konidaris, G.D., and Barto, A. Learning parameterized skills. In *Proceedings of the Twenty Ninth International Conference on Machine Learning* (2012), pp. 1679–1686.

[18] Dearden, R., Friedman, N., and Andre, D. Model based bayesian exploration. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence* (1999), pp. 150–159.

[19] Deisenroth, M.P., Englert, P., Peters, J., and Fox, D. Multi-task policy search for robotics. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation* (2014).

[20] Drucker, H., Burges, C., Kaufman, L., Smola, A., and Vapnik, V. Support vector regression machines. In *Advances in Neural Information Processing Systems 9* (1997), MIT Press, pp. 155–161.

[21] Duff, M.O. Design for an optimal probe. In *Proceedings of the 20th International Conference on Machine learning* (2003), AAAI Press, pp. 131–138.

[22] Epshteyn, A., Vogel, A., and DeJong, G. Active reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning* (2008), Omnipress, pp. 296–303.

[23] Ericsson, K. *The influence of experience and deliberate practice on the development of superior expert performance.* Cambridge University Press, 2006, ch. 13, pp. 685–708.

[24] Harlow, H. F. Learning and satiation of response in intrinsically motivated complex puzzle performance by monkeys. *Journal of Comparative and Physiological Psychology 43* (1950), 289–294.

[25] Hart, S., Sen, S., and Grupen, R.A. Generalization and transfer in robot control. In *Proceedings of the Eighth International Conference on Epigenetic Robotics* (2008).

[26] Hausknecht, M., and Stone, P. Learning powerful kicks on the Aibo ERS-7: The quest for a striker. In *RoboCup-2010: Robot Soccer World Cup XIV*, vol. 6556 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2011, pp. 254–65.

[27] Hoffmann, H., Petkos, G., Bitzer, S., and Vijayakumar, S. Sensor-assisted adaptive motor control under continuously varying context. In *Proceedings of the Tenth International Conference on Informatics in Control, Automation, and Robotics* (2007).

[28] Huber, M., and Grupen, R.A. Learning to coordinate controllers-reinforcement learning on a control basis. In *Proceedings of the Fifteenth International Joint Conference on Artificial intelligence* (1997), pp. 1366–1371.

[29] Jordan, M. Rumelhart, D. Forward models: Supervised learning with a distal teacher. *Cognitive Science 16* (1992), 307–354.

[30] Kearns, M.J., and Singh, S. Near-optimal reinforcement learning in polynominal time. In *Proceedings of the 15th International Conference on Machine Learning* (1998), Morgan Kaufmann, pp. 260–268.

[31] Kober, J., and Peters, J. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems 21* (2008), pp. 849–856.

[32] Kober, J., and Peters, J. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine 17*, 2 (2010), 55–62.

[33] Kober, J., Wilhelm, A., Oztop, .E, and Peters, J. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots* (2012), 1–19.

[34] Kohl, N., and Stone, P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2004).

[35] Kolter, J.Z., and Ng, A. Near-Bayesian exploration in polynomial time. In *Proceedings of the 26th International Conference on Machine Learning* (2009), Omnipress, pp. 513–520.

[36] Konidaris, G.D., and Barto, A.G. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (2007), pp. 895–900.

[37] Konidaris, G.D., and Barto, A.G. Efficient skill learning using abstraction selection. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence* (2009).

[38] Konidaris, G.D., and Barto, A.G. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22* (2009), pp. 1015–1023.

[39] Konidaris, G.D., Osenstoski, S., and Thomas, P. Value function approximation in reinforcement learning using the fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence* (2011), pp. 380–385.

[40] Konidaris, G.D., Scheidwasser, I., , and Barto, A.G. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research 13* (2012), 1333–1371.

[41] Kroemer, O., Detry, R., Piater, J., and Peters, J. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems 58*, 9 (2010), 1105–1116.

[42] Kuindersma, S., Grupen, R., and Barto, A. Variational Bayesian optimization for runtime risk-sensitive control. In *Robotics: Science and Systems VIII (RSS)* (Sydney, Australia, 2012).

[43] Levine, S., Popovic, Z., and Koltun, V. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems 24* (2011), pp. 19–27.

[44] Liu, Y., and Stone, P. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings to the Twenty-First National Conference on Artificial Intelligence* (2006), pp. 415–420.

[45] Mahadevan, S., Giguere, S., , and Jacek, N. Basis adaptation for sparse nonlinear reinforcement learning. In *Proceedings of the Twenty-Seventh Conference on Artificial Intelligence* (2013).

[46] Mannor, S., Rubinstein, R., and Gat, Y. The cross entropy method for fast policy search. In *Proceedings of the Twentieth International Conference on Machine Learning* (2003), pp. 512–519.

[47] McGovern, A., and Barto, A.G. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning* (2001), pp. 361–368.

[48] Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks 23*, 8-9 (2010), 1125–1134.

[49] Moore, A.W., and Atkeson, C.G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning 13* (1993), 103–130.

[50] Nemec, B., Forte, D., Vuga, R., Tamosiunaite, M., Wörgötter, F., and Ude, A. Applying statistical generalization to determine search direction for reinforcement learning of movement primitives. In *Proceedings of the Twelfth IEEE-RAS International Conference on Humanoid Robots* (2012), pp. 65–70.

[51] Neumann, G., Daniel, C., Kupcsik, A, Deisenroth, M., and Peters, J. Information-theoretic motor skill learning. In *Proceedings of the AAAI Workshop on Intelligent Robotic Systems* (2013).

[52] Ng, A.Y., and Russell, S.J. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning* (2000), pp. 663–670.

[53] Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. Learning and generalization of motor skills by learning from demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2009).

[54] Precup, D. *Temporal Abstraction in Reinforcement Learning.* PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, 2000.

[55] Rajendran, S., and Huber, M. Autonomous identification, categorization and generalization of policies based on task type. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (2011), pp. 1333–1339.

[56] Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[57] Rottmann, A., and Burgard, W. Learning non-stationary system dynamics online using gaussian processes. In *Proceedings of the Thirty-second Symposium of the German Association for Pattern Recognition* (2010), pp. 192–201.

[58] Ryan, R., and Deci, E. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology 25*, 1 (2000), 54–67.

[59] Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. Learning movement primitives. In *Proceedings of the Eleventh International Symposium on Robotics Research* (2004), Springer.

[60] Schmidhuber, J. Adaptive confidence and adaptive curiosity. Tech. Rep. FKI-149-91, Technische Universitat Munchen, 1991.

[61] Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats* (1991), pp. 222–227.

[62] Silver, D., Bagnell, J., and Stentz, A. Active learning from demonstration for robust autonomous navigation. In *Proceedings of 2012 IEEE International Conference on Robotics and Automation* (2012).

[63] Singh, S., Barto, A.G., and Chentanez, N. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005, pp. 1281–1288.

[64] Snoek, J., Larochelle, H., and Adams, R. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25* (2012), pp. 2951–2959.

[65] Soni, V., and Singh, S. Reinforcement learning of hierarchical skills on the Sony Aibo robot. In *Proceedings of the Fifth International Conference on Development and Learning* (2006).

[66] Stout, A., and Barto, A.G. Competence progress intrinsic motivation. In *Proceedings of the Ninth International Conference on Development and Learning* (2010), pp. 257–262.

[67] Stout, A., Konidaris, G.D., and Barto, A.G. Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning. In *Proceedings of the AAAI Spring Symposium on Developmental Robotics* (2005).

[68] Strehl, A.L., Li, L., Wiewiora, E., Langford, J., and Littman, M. PAC model-free reinforcement learning. In *Proceedings of the 23rd International Conference on Machine learning* (2006), pp. 881–888.

[69] Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. Learning compact parameterized skills with a single regression. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots* (2013), pp. 1–7.

[70] Stulp, F., and Sigaud, O. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning* (2012).

[71] Stulp, F., and Sigaud, O. Policy improvement: Between black-box optimization and episodic reinforcement learning. In *Journées Francophones Planification, Décision, et Apprentissage pour la conduite de systèmes* (2013).

[72] Sutton, R., Mcallester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12* (2000), MIT Press, pp. 1057–1063.

[73] Sutton, R., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence 112* (1999), 181–211.

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

[75] Taylor, M., Kuhlmann, G., and Stone, P. Autonomous transfer for reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (2008).

[76] Taylor, M., and Stone, P. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning* (2007).

[77] Taylor, M., Stone, P., and Liu, Y. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research 8*, 1 (2007), 2125–2167.

[78] Tenenbaum, J., de Silva, V., and Langford, J. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323.

[79] Theodorou, E., Buchli, J., and Schaal, S. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research 11* (2010), 3137–3181.

[80] Thrun, S.B. Efficient exploration in reinforcement learning. Tech. Rep. CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[81] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research 6* (2005), 1453–1484.

[82] Van Vaerenbergh, S., Santamaría, I., and Lázaro-Gredilla, M. Estimation of the forgetting factor in kernel recursive least squares. In *Proceedings of the IEEE International Workshop On Machine Learning For Signal Processing* (2012).

[83] Watkins, Christopher J. C. H., and Dayan, Peter. Q-learning. *Machine Learning 8*, 3 (1992), 279–292.

[84] Weiss, D., and Taskar, B. Structured prediction cascades. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), vol. 9, pp. 916–923.

[85] White, R. Motivation reconsidered: The concept of competence. *Psychological Review 66*, 5 (1959), 297–333.

[86] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8*, 3-4 (1992), 229–256.