Acquiring Diverse Robot Skills via Maximum Entropy Deep Reinforcement Learning

Tuomas Haarnoja

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-176
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-176.html

December 14, 2018

Acquiring Diverse Robot Skills via Maximum Entropy Deep Reinforcement Learning

by

Tuomas Haarnoja

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Fall 2018

Acquiring Diverse Robot Skills
via Maximum Entropy Deep Reinforcement Learning

Abstract

Acquiring Diverse Robot Skills
via Maximum Entropy Deep Reinforcement Learning

by

Tuomas Haarnoja

Doctor of Philosophy in Computer Science

University of California, Berkeley

Assistant Professor Sergey Levine, Chair
Professor Pieter Abbeel, Chair

In this thesis, we study how maximum entropy framework can provide efficient deep reinforcement learning (deep RL) algorithms that solve tasks consistently and sample efficiently. This framework has several intriguing properties. First, the optimal policies are stochastic, improving exploration and preventing convergence to local optima, particularly when the objective is multimodal. Second, the entropy term provides regularization, resulting in more consistent and robust learning when compared to deterministic methods. Third, maximum entropy policies are composable, that is, two or more policies can be combined, and the resulting policy can be shown to be approximately optimal for the sum of the constituent task rewards. And fourth, the view of maximum entropy RL as probabilistic inference provides a foundation for building hierarchical policies that can solve complex and sparse reward tasks. In the first part, we will devise new algorithms based on this framework, starting from soft Q-learning that learns expressive energy-based policies, to soft actor-critic that provides simplicity and convenience of actor-critic methods, and ending with automatic temperature adjustment scheme that practically eliminates the need for hyperparameter tuning, which is a crucial feature for real-world applications where tuning of hyperparameters can be prohibitively expensive. In the second part, we will discuss extensions enabled by the inherent stochasticity of maximum entropy polices, including compositionality and hierarchical learning. We will demonstrate the effectiveness of the proposed algorithms on both simulated and real-world robotic manipulation and locomotion tasks.

To my wife
and
to my parents.

Contents

Acknowledgments

The last five years have taught me more than what I could have ever imagined, both professionally and personally. This journey has only been possible due to the abundant support from countless people around me. I have been extremely fortunate to be advised by two of the very best of their field, Sergey Levine and Pieter Abbeel. I had little, if any, experience in machine learning when I first contacted Pieter, and yet he was kind and supportive enough to let me join his team, and throughout the years, he helped me to pivot towards becoming an AI researcher. I am tremendously grateful to Sergey Levine, who helped to refine my research goals further and who always (quite literally) found the time to guide and encourage me, from the initial brainstorming to the preparation of the final manuscript.

I am grateful to my collaborators—both at UC Berkeley and at Google Brain—most importantly Aurick Zhou and Kristian Hartikainen, who I have worked most closely with, and other collaborators, including Anurag Ajay, Murtaza Dalal, Jasmine Deng, Sehoon Ha, Vitchyr Pong, Jie Tan, Haoran Tang, and George Tucker, whose contribution to my research was priceless. I would also like to thank all the people with whom I have had a chance to exchange ideas, validate my thoughts, and discuss research in general: Pulkit Agrawal, Roberto Calandra, Ignasi Clavera, Benjamin Eysenbach, Carlos Florensa, Pim de Haan, Danijar Hafner, Nicolas Heess, Ethan Holly, Young Geng, Abhishek Gupta, Gregory Kahn, Aviral Kumar, Vikash Kumar, Aldo Pacchiano, Adam Stooke, Vincent Vanhoucke, and countless others, including the members of RAIL and RLL, and the researchers I have encountered at UC Berkeley, Google, and various conferences. I would also like to thank Roy Fox for teaching me how to organize a research workshop.

This journey would not have been possible without my former supervisors, Kari Tammi and José Luis Peralta, who prepared me and provided tremendous support for my effort to become a graduate student at UC Berkeley. I am thankful to various institutions for funding during my first years, including the Fulbright-Technology Industries of Finland program, Tutkijat Maailmalle program, Walter Ahlström Foundation, and Jenny and Antti Wihuri Foundation, as well Berkeley DeepDrive and Siemens after joining first Pieter's team and later Sergey's team.

I am thankful to Thanh Do, Lily Hu, Drew Sabelhaus, and Matt Wright, and other friends who I met in my first year and who helped me to adapt to the new culture, and to the friends with whom I have done countless runs and hikes, including Dara Bahri, Frederik Ebert, Carlos Florensa, Pim de Haan, Nils Lundt, Teemu Pitkänen, Avi Singh, and Adam Stooke. I would like to thank my support team in Finland: Jouko Kinnari, Juuso Lehtonen, Ville Liimatainen, Juan Prajogo, and others, who persistently go for a drink with me—twice a year—and share the latest rumors and updates, and help me not forget my origins. I would like to thank my closest family and relatives, and the folks at Kustavi island, which has almost become my second home during my visits to Finland.

I am deeply grateful to my wife for bearing with me throughout the times I have spent afar, for taking care of our home, for representing us, and for not giving up.

Chapter 1

Introduction

1.1 Deep Reinforcement Learning for Robotics

Deep reinforcement learning (deep RL) has emerged as a promising direction for autonomous acquisition of complex behaviors (Mnih et al., 2015; Silver et al., 2016), due to its ability to process complex sensory input (Jaderberg et al., 2017) and to acquire elaborate behavior skills using general-purpose neural network representations (Levine et al., 2016). These properties make deep RL particularly appealing in the domain of robotic manipulation and locomotion, where manual controller design can be difficult and highly task and robot specific. If we can learn to solve the tasks from scratch directly in the real world, we can in principle acquire controllers that are ideally adapted to each robot and environment, potentially achieving better performance, energy efficiency, and robustness. Deep reinforcement learning has been used extensively to learn locomotion policies in simulation (Heess et al., 2017; Xie et al., 2018; Peng et al., 2016; Berseth et al., 2018) and even transfer them to real-world robots (Tan et al., 2018), but this inevitably incurs some loss of performance due to discrepancies in the simulation, and requires extensive manual modeling. Attempts to apply deep reinforcement learning directly in the real world has proven challenging and has been limited to simple manipulation tasks (Gu et al., 2016; Večerík et al., 2017; Mahmood et al., 2018), locomotion tasks with inherently stable robots (Ha et al., 2018), low-dimensional parameterizations (Kohl & Stone, 2004; Calandra et al., 2016; Rai et al., 2017), or both (Tedrake et al., 2005).

Applying end-to-end deep reinforcement learning with general-purpose function approximators is complicated by two major challenges. The first is the large sample requirements of many deep reinforcement learning algorithms, which might require tens of thousands of trials in the real world, and the second, more subtle challenge is the severe sensitivity many of these methods have to hyperparameters settings (Henderson et al., 2018). In simulation, hyperparameters can be tuned in parallel, but in the real world, this requires multiple distinct training runs, further exacerbating the already severe sample complexity challenges. Many robotic systems, especially legged robots that can fall and damage themselves, and manipulators with mechanical gears, simply cannot survive so many repeated trials, especially under the control of a suboptimal and partially trained policy.

One cause for the poor sample efficiency of deep RL methods is on-policy learning: some of the most commonly used deep RL algorithms, such as TRPO (Schulman et al., 2015a), PPO (Schulman et al., 2017b) or A3C (Mnih et al., 2016), require new samples to be collected for each gradient step. This quickly becomes extravagantly expensive, as the number of gradient steps and samples per step needed to learn an effective policy increases with task complexity. Off-policy algorithms aim to reuse past experience. This is not directly feasible with conventional policy gradient formulations, but is relatively straightforward for Q-learning based methods (Mnih et al., 2015). Unfortunately, the combination of off-policy learning and high-dimensional, nonlinear function approximation with neural networks presents a major challenge for stability and convergence (Bhatnagar et al., 2009). This challenge is further exacerbated in continuous state and action spaces, where a separate actor network is often used to perform the maximization in Q-learning. A commonly used algorithm in such settings, deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), provides for sample-efficient learning but can be challenging to use due to its brittleness and hyperparameter sensitivity (Duan et al., 2016; Henderson et al., 2018). In this thesis, we study how maximum entropy deep RL can help mitigate these problems by enabling algorithms that are robust against noise in the learning process and variations in the hyperparameters, and lead to practical extensions that promote reusability of already learned skills.

1.2 Reinforcement Learning with Stochastic Policies

Deep reinforcement learning methods can be used to optimize deterministic (Lillicrap et al., 2015) and stochastic (Schulman et al., 2015a; Mnih et al., 2016) policies. However, most deep RL methods operate on the conventional deterministic notion of optimality, where the optimal solution, at least under full observability, is always a deterministic policy (Sutton & Barto, 1998). Even though often times we prefer a deterministic policy at convergence, a stochastic policy is desirable for exploration. This exploration is typically attained heuristically, for example, by injecting noise (Silver et al., 2014; Lillicrap et al., 2015; Mnih et al., 2015) or initializing a stochastic policy with high entropy (Kakade, 2002; Schulman et al., 2015a; Mnih et al., 2016) as opposed to optimizing as part of the objective function and learning algorithm. In some cases, we might actually prefer to learn stochastic behaviors. Learning a stochastic policy can lead to structured exploration and can be easily adapted to new situations (as we discuss in Chapter 3), can help derive better algorithms that exhibit stable and consistent learning (Chapter 4 and Chapter 5), are composable (Chapter 6), and provide a foundation for hierarchical reinforcement learning (Chapter 7). However, in order to learn stochastic policies, we must define an objective that promotes stochasticity.

In which cases is a stochastic policy actually the optimal solution? As discussed in prior work, a stochastic policy emerges as the optimal answer when we consider the connection between optimal control and probabilistic inference (Todorov, 2008). While there are multiple instantiations of this framework (Kappen, 2005; Todorov, 2007; Ziebart et al., 2008; Neumann, 2011; Rawlik et al., 2012; Fox et al., 2016), they typically include the cost or reward function as an additional factor in a probabilistic graphical model and infer the conditional distribution over actions conditioned on states and optimality. The solution can be shown to optimize an entropy-augmented reinforcement learning objective or to correspond to the solution to a maximum entropy learning problem (Levine, 2018). Intuitively, framing control as inference produces policies that aim to capture not only the single deterministic behavior that has the lowest cost, but the entire range of low-cost behaviors, explicitly maximizing the entropy of the corresponding policy. Instead of learning the best way to perform the task, the resulting policies try to learn all of the ways of performing the task. In this thesis, we propose sample efficient and robust reinforcement learning algorithms based on this framework. Our ultimate goal is to devise algorithms that are sufficiently sample efficient and robust to be applicable to real-world robotic tasks.

1.3 Problem Setting

We address learning of maximum entropy policies in continuous action spaces. Our reinforcement learning problem can be defined as a policy search in an a Markov decision process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space; the state transition probability $p: \ \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$ represents the probability density of the next state $\mathbf{s}_{t+1} \in \mathcal{S}$ given the current state $\mathbf{s}_t \in \mathcal{S}$ and action $\mathbf{a}_t \in \mathcal{A}$; and $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{\min}, r_{\max}]$ represents the reward function. We use $\rho_\pi(\tau)$ to denote the density over trajectories $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, ...)$ induced by a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$, and we overload the notation and use $\rho_\pi(\mathbf{s}_t)$ and $\rho_\pi(\mathbf{s}_t, \mathbf{a}_t)$ to denote the state and state-action marginals, respectively.

Our goal is to learn a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$. We can define the standard reinforcement learning objective in terms of the above quantities as the expected return $\sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r(\mathbf{s}_t, \mathbf{a}_t)\right]$. The maximum entropy objective (see e.g. (Ziebart, 2010)) is more general, and it augments the standard objective with an entropy term, such that the optimal policy aims to maximize its entropy at each visited state:

$$\pi^* = \arg\max_\pi \sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t))\right], \tag{1.1}$$

where α is the temperature parameter that determines the relative importance of the entropy term against the reward, and thus controls the stochasticity of the optimal policy. The policy is incentivized to explore more widely, while giving up on clearly unpromising avenues, and can capture multiple modes of near-optimal behavior: in problem settings where multiple actions seem equally attractive, the policy will commit equal probability mass to those actions. The maximum entropy objective differs from the standard maximum expected reward objective used in conventional reinforcement learning, but the conventional objective can be recovered in the limit as $\alpha \rightarrow 0$. If we wish to extend either the conventional or the maximum entropy RL objective to infinite horizon problems, it is convenient to also introduce a discount factor γ to ensure that the sum of expected rewards (and entropies) is finite, but writing down the precise objective that is optimized when using the discount factor is non-trivial (Thomas, 2014). We defer the definition of the discounted objective to Appendix A, but we will consider the discounted objective in the following chapters.

1.4 Overview and Contributions

This thesis concludes the research contributions of five previously published papers. The organization does not strictly follow the order in which the works were published, but instead some of the material has been rearranged to give the thesis a more coherent structure and to improve readability. We have split the thesis in two parts. In Part I, we introduce new algorithms for learning maximum entropy policies, yielding state-of-the-art performance without the need for exhaustive hyperparameter tuning. In Part II, we discuss extensions to the maximum entropy framework that can further improve sample complexity by reuse of skills. Specifically, the main contributions are the following:

Part I

- In Chapter 3, we show how we can extend soft Q-learning, a reinforcement algorithm that optimizes the maximum entropy objective, to continuous actions. The prior work (Todorov, 2007; Toussaint, 2009; Rawlik et al., 2012) has considered only discrete action spaces. The optimal policy has an energy-based form, and thus sampling from it becomes intractable in continuous and large action spaces. Our solution uses approximate inference based on Stein variational gradient descent (Wang & Liu, 2016) and can learn expressive behaviors, resulting in multimodal exploration and policies that can be easily adapted to new environments. We show that soft Q-learning can successfully acquire real-world manipulation skills for tasks that prior methods fail to solve. We also present a proof equating soft Q-learning and policy gradients with entropy regularization (concurrently with Schulman et al. (2017a)). The soft Q-learning algorithm for continuous actions was previously published in (Haarnoja et al., 2017) and the real-world training results in (Haarnoja et al., 2018b).

- In Chapter 4, we propose a new algorithm, soft actor-critic, for learning maximum entropy policies restricted to parameterized families of tractable policy distributions. The main challenge with soft Q-learning is it makes an implicit assumption that the policy can represent any distribution, and consequently, the optimal policy is energy based. Use of a tractable policy, for example, Gaussian, would prevent soft Q-learning from converging to the optimal solution. Soft actor-critic can handle additional optimization constraints while provably converges to the optimal solution from any given family of distributions. The algorithm yields state-of-the-art performance in simulated locomotion tasks. This work was presented in (Haarnoja et al., 2018c).

- In Chapter 5, we propose an automatic temperature adjustment scheme for maximum entropy RL algorithms. One of the major challenges of maximum entropy RL is the choice of the right temperature parameter, which is used to balance between exploration and exploitation. Our approach is to recast the problem as a maximization of the expected return subject to a constraint on the expected entropy. We can show that the dual of the constrained optimization problem is equivalent to learning a maximum entropy policy with an temperature corresponding to the dual variable. In comparison to the standard approach, which uses a fixed entropy and anneals it towards zero in the course of training, our approach constraints the expected entropy over the visited states, and the learning algorithm can optimally distribute the entropy budget across the visited states. The experimental validation indicates that the automatic tuning scheme yields results that are on a par with manually tuned, fixed temperature, and hence effectively eliminates the need for manual tuning. We demonstrate the effectiveness of the method by learning locomotion policies on a real-world quadrupedal platform from scratch. This work was conducted at Google Brain in Mountain View, California as an internship project and a publication is currently in preparation (Haarnoja et al., 2019).

Part II

- In Chapter 6, we show that maximum entropy policies for a set of tasks can be combined to form a new policy that is approximately optimal with respect to the constituent tasks. We derive a bound for the suboptimality of such policies, and demonstrate compositionality in both simulated and real-world manipulation tasks. Prior work has considered the "or" compositionality (Todorov, 2009), where the optimal strategy is to choose the action based on the most rewarding task (i.e. solve only the easiest task). On the contrary, we consider the "and" compositionality, where the solution is to act optimally with respect to both tasks at the same time. This kind of compositionality is common in robotics, where the end goal can typically be expressed as a sum of multiple rewards, corresponding to, for example, avoidance of an obstacle and reaching a target object. This work was previously published in (Haarnoja et al., 2018b).

- In Chapter 7, we extend the probablistic view of maximum entropy reinforcement learning to hierarchical policies. Specifically, maximum entropy policy can be viewed as an action distribution conditioned on a state and optimality variables. We additionally condition the policy on a latent variable. If we augment the graphical model with a learned policy, we can view the latent as a new input to the system, and infer the optimal latent analogously to the actions. This procedure can be repeated arbitrarily many times, leading to a sequence of latents, which—if viewed as intermediate actions—correspond to a hierarchical multi-level policy. We represent the latent space policies using normalizing flows (Dinh et al., 2016), which have been previously used for generative modeling of images. We demonstrate hierarchical learning on a simulated sparse navigation task, and the results indicate substantial improvement over prior methods in terms of sample complexity. This work was previously presented in (Haarnoja et al., 2018a).

Finally, in Chapter 8, we conclude and discuss potential avenues for the future research.

Chapter 2

Related Work

2.1 Maximum Entropy Reinforcement Learning

Maximum entropy policies emerge as the solution when we cast optimal control as probabilistic inference. In the case of linear-quadratic systems, the mean of the maximum entropy policy is exactly the optimal deterministic policy (Todorov, 2008), which has been exploited to construct practical path planning methods based on iterative linearization and probabilistic inference techniques (Toussaint, 2009). In discrete state spaces, the maximum entropy policy can be obtained exactly. This has been explored in the context of linearly solvable MDPs (Todorov, 2007) and, in the case of inverse reinforcement learning (MaxEnt IRL) (Ziebart et al., 2008). In continuous systems and continuous time, path integral control studies maximum entropy policies and maximum entropy planning (Kappen, 2005) and recently proposed path consistency learning learns a Gaussian policy by minimizing the soft Bellman consistency (Nachum et al., 2017, 2018). In contrast to these prior methods, this thesis focuses on extending the maximum entropy policy search framework to high-dimensional continuous spaces and highly multimodal objectives.

A number of related methods have also used maximum entropy policy optimization as an intermediate step for optimizing policies under a standard expected reward objective (Peters et al., 2010; Neumann, 2011; Rawlik et al., 2012; Fox et al., 2016). Among these, the work of Rawlik et al. (2012) resembles ours in that it also makes use of a temporal difference style update to a soft Q-function. However, unlike this prior work, soft Q-learning (SQL) uses general-purpose energy functions and approximate sampling, rather than analytically normalizable distributions. A recent work (Liu et al., 2017) also considers an entropy regularized objective, though the entropy is on policy parameters, not on sampled actions. Thus the resulting policy may not represent an arbitrarily complex multimodal distribution with a single parameter. The form of our sampler resembles the stochastic networks proposed in recent work on hierarchical learning (Florensa et al., 2017). However this prior work uses a task-specific reward bonus system to encourage stochastic behavior, while our approach is derived from optimizing a general maximum entropy objective.

A closely related concept to maximum entropy policies is Boltzmann exploration, which uses the exponential of the standard Q-function as the probability of an action (Kaelbling et al., 1996). A number of prior works have also explored representing policies as energy-based models, with the Q-value obtained from an energy model such as a restricted Boltzmann machine (RBM) (Sallans & Hinton, 2004; Elfwing et al., 2010; Otsuka et al., 2010; Heess et al., 2013). Although these methods are closely related, they have not, to our knowledge, been extended to the case of deep network models, have not made extensive use of approximate inference techniques, and have not been demonstrated on the complex continuous tasks. More recently, O'Donoghue et al. (2016) drew a connection between Boltzmann exploration and entropy-regularized policy gradient, though in a theoretical framework that differs from maximum entropy policy search: unlike the full maximum entropy framework, the approach of O'Donoghue et al. (2016) only optimizes for maximizing entropy at the current time step, rather than planning for visiting future states where entropy will be further maximized.

2.2 Actor Critic Methods

Our soft actor-critic (SAC) algorithm incorporates three key ingredients: an actor-critic architecture with separate policy and Q-value networks, an off-policy formulation that enables reuse of previously collected data for efficiency, and entropy maximization to enable stability and exploration. Actor-critic algorithms are typically derived starting from policy iteration, which alternates between policy evaluation—computing the value function for a policy—and policy improvement—using the value function to obtain a better policy (Barto et al., 1983; Sutton & Barto, 1998). In large-scale reinforcement learning problems, it is typically impractical to run either of these steps to convergence, and instead the value function and policy are optimized jointly. In this case, the policy is referred to as the actor, and the value function as the critic. Many actor-critic algorithms build on the standard, on-policy policy gradient formulation to update the actor (Peters & Schaal, 2008), and many of them also consider the entropy of the policy, but instead of maximizing the entropy, they use it as an regularizer (Schulman et al., 2017b, 2015a; Mnih et al., 2016; Gruslys et al., 2017). On-policy training tends to improve stability but results in poor sample complexity.

There have been efforts to increase the sample efficiency while retaining robustness by incorporating off-policy samples and by using higher order variance reduction techniques (O'Donoghue et al., 2016; Gu et al., 2017). However, fully off-policy algorithms still attain better efficiency. A particularly popular off-policy actor-critic method, deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), which is a deep variant of the deterministic policy gradient (Silver et al., 2014) algorithm, uses a Q-function estimator to enable off-policy learning, and a deterministic actor that maximizes this Q-function. As such, this method can be viewed both as a deterministic actor-critic algorithm and an approximate Q-learning algorithm. Unfortunately, the interplay between the deterministic actor network and the Q-function typically makes DDPG difficult to stabilize and brittle to hyperparameter settings (Duan et al., 2016; Henderson et al., 2018). As a consequence, it is difficult to extend DDPG to complex, high-dimensional tasks, and on-policy policy gradient methods still tend to produce the best results in such settings (Gu et al., 2017). Soft actor-critic instead combines off-policy actor-critic training with a stochastic actor, and further aims to maximize the entropy of this actor with an entropy maximization objective. We find that this actually results in a considerably more stable and scalable algorithm that, in practice, exceeds both the efficiency and final performance of DDPG. A similar method can be derived as a zero-step special case of stochastic value gradients (SVG(0)) (Heess et al., 2015). However, SVG(0) differs from our method in that it optimizes the standard maximum expected return objective.

Recently, several papers have noted the connection between Q-learning and policy gradient methods in the framework of maximum entropy learning (O'Donoghue et al., 2016; Haarnoja et al., 2017; Nachum et al., 2017; Schulman et al., 2017a). While most of the prior model-free works assume a discrete action space, Nachum et al. (2018) approximate the maximum entropy distribution with a Gaussian and soft Q-learning (Chapter 3) with a sampling network trained to draw samples from the optimal policy. Although soft Q-learning has a Q-function and an actor network, it is not a true actor-critic algorithm: the Q-function is estimating the optimal soft Q-function, and the actor does not directly affect the soft Q-function except through the data distribution. Hence, the actor is motivated as an approximate sampler, rather than the actor in an actor-critic algorithm. Crucially, the convergence of this method hinges on how well this sampler approximates the true posterior. In contrast, we prove that soft actor-critic converges to the optimal policy from a given policy class, regardless of the policy parameterization. Furthermore, these prior maximum entropy methods generally do not exceed the performance of state-of-the-art off-policy algorithms, such as DDPG, when learning from scratch, though they may have other benefits, such as improved exploration and ease of fine-tuning as we show in Chapter 3. In our experiments, we demonstrate that our soft actor-critic algorithm does in fact exceed the performance of prior state-of-the-art off-policy deep RL methods.

2.3 Reinforcement Learning in the Real-World

One of the crucial choices in applying RL for robotic manipulation is the choice of representation. Policies based on dynamic movement primitives and other trajectory-centric representations, such as splines, have been particularly popular, due to the ease of incorporating demonstrations, stability, and relatively low dimensionality (Ijspeert et al., 2003, 2013; Peters & Schaal, 2006, 2008). However, trajectory-centric policies are limited in their expressive power, particularly when it comes to integrating rich sensory information and reacting continuously to the environment. For this reason, recent works have sought to explore more expressive representations, including deep neural networks, at the cost of higher sample complexity.

A number of prior works have sought to address the increased sample complexity by employing model-based methods. For example, guided policy search (Levine et al., 2016) learns a model-based teacher that supervises the training of a deep policy network. Other works leverage simulation: Ghadirzadeh et al. (2017) considers vision-based manipulation by training perception and behavior networks in simulation and learns only a low-dimensional intermediate layer with real-world interaction. Some works learn vision-based policies completely in simulation and then transfers them into real world (Zhang et al., 2015; Sadeghi & Levine, 2017; Tobin et al., 2017; Andrychowicz et al., 2017). For this approach to work, the simulator needs to model the system accurately and there needs to be significant variation in the appearance of the synthetic images in order to handle the unavoidable domain-shift between the simulation and the real-world (Sadeghi & Levine, 2017; Tobin et al., 2017; James et al., 2017; Bateux et al., 2017). Another common approach to make reinforcement learning algorithms more sample efficient is to learn from demonstrations (Guenter et al., 2007; Pastor et al., 2009; Theodorou et al., 2010; Večerík et al., 2017), but this comes at the cost of additional instrumentation and human supervision.

Perhaps the most closely related recent work that aims to apply model-free deep RL algorithms to real-world robotic manipulation include the work by Gu et al. (2016) who used normalized advantage functions (NAF) to learn door opening and reaching by parallelizing across multiple real-world robots, and Večerík et al. (2017) who extended DDPG to real-world robotic manipulation by including example demonstrations. Our experiments show that soft Q-learning (SQL) can learn real-world manipulation more proficiently and with fewer samples than methods such as DDPG and NAF, without requiring demonstrations, simulated experience, or additional supervision.

2.4 Composable Reinforcement Learning

Prior works (Todorov, 2007, 2008) have studied composition in the context of soft optimality, but typically in a different framework: instead of considering composition where the reward functions of constituent skills are added (i.e., do X *and* Y), they considered settings where a soft maximum ("log-sum-exp") over the reward functions is used as the reward for the composite skill (i.e., do X *or* Y). We argue that the former is substantially more useful than the latter for robotic skills, since it allows us to decompose a task into multiple objectives that all need to be satisfied.

2.5 Hierarchical Reinforcement Learning

A number of prior works have explored how reinforcement learning can be cast in the framework of probabilistic inference (Kappen, 2005; Todorov, 2007; Ziebart et al., 2008; Toussaint, 2009; Peters et al., 2010; Neumann, 2011). The approach we take in Chapter 7 is based on a formulation of reinforcement learning as inference in a graphical model (Ziebart et al., 2008; Toussaint, 2009; Levine, 2014). Building on this graphical model interpretation of reinforcement learning also makes it natural for us to augment the policy with latent variables. While several prior works have sought to combine maximum entropy policies with learning of latent spaces (Haarnoja et al., 2017; Hausman et al., 2018) and even with learning hierarchies in small state spaces (Saxe et al., 2017), to our knowledge, our method is the first to extend this mechanism to the setting of learning hierarchical policies with deep RL in continuous domains.

Prior frameworks for hierarchical learning are often based on either options or contextual policies. The options framework (Sutton et al., 1999) combines low-level option policies with a top-level policy that invokes individual options, whereas contextual policies (Kupcsik et al., 2013; Schaul et al., 2015; Heess et al., 2016) generalize options to continuous goals. One of the open questions in both options and contextual policy frameworks is how the base policies should be acquired. In some situations, a reasonable solution is to resort to domain knowledge and design a span of subgoals manually (Heess et al., 2016; Kulkarni et al., 2016; MacAlpine & Stone, 2018). Another option is to train the entire hierarchy end-to-end (Bacon et al., 2017; Vezhnevets et al., 2017; Daniel et al., 2012). While the end-to-end training scheme provides generality and flexibility, it is prone to learning degenerate policies that exclusively use a single option, losing much of the benefit of the hierarchical structure (Bacon et al., 2017). To that end, the option-critic (Bacon et al., 2017) adopts a standard entropy regularization scheme ubiquitous in policy gradient methods (Mnih et al., 2016; Schulman et al., 2015a), Florensa et al. propose maximizing the mutual information of the top-level actions and the state distribution, and Daniel et al. bound the mutual information of the actions and top-level actions. Our method also uses entropy maximization to obtain diverse base policies, but in contrast to prior methods, our sub-policies are invertible and parameterized by continuous latent variables. The higher levels can thus undo any lower level transformation, and the lower layers can learn independently, allowing us to train the hierarchies in bottom-up layerwise fashion. Unlike prior methods, which use structurally distinct higher and lower layers, all layers in our hierarchies are structurally generic, and are trained with exactly the same procedure.

Part I

Algorithms

Chapter 3

Soft Q-Learning

In this chapter, we first describe a value-based approach to maximum entropy RL. We then propose a method for learning expressive energy-based policies for continuous states and actions, resulting into a new soft Q-learning (SQL) algorithm that expresses the optimal policy via a Boltzmann distribution. We use the recently proposed amortized Stein variational gradient descent to learn a stochastic sampling network that approximates samples from this distribution. The benefits of the proposed algorithm include improved exploration and better transferability of skills between tasks, which we confirm in simulated experiments with swimming and walking robots. We also demonstrate how soft Q-learning can solve real-world robotic tasks from scratch more efficiently than the prior methods and that the resulting policies are surprisingly robust to perturbations.

3.1 Overview

Solving maximum entropy stochastic policy learning problems in the general case is challenging. A number of methods have been proposed, including Z-learning (Todorov, 2007), maximum entropy inverse RL (Ziebart et al., 2008), approximate inference using message passing (Toussaint, 2009), Ψ-learning (Rawlik et al., 2012), and G-learning (Fox et al., 2016), as well as more recent proposals in deep RL such as PGQ (O'Donoghue et al., 2016) and path consistency learning (Nachum et al., 2017), but these generally operate either on simple tabular representations, which are difficult to apply to continuous or high-dimensional domains, or employ a simple parametric representation of the policy distribution, such as a conditional Gaussian. Therefore, although the policy is optimized to perform the desired skill in many different ways, the resulting distribution is typically very limited in terms of its representational power, even if the parameters of that distribution are represented by an expressive function approximator, such as a neural network. In our method, we formulate a stochastic policy as a (conditional) energy-based model (EBM), with the energy function corresponding to the "soft" Q-function obtained when optimizing the maximum entropy objective. In high-dimensional continuous spaces, sampling from this policy, just as with any

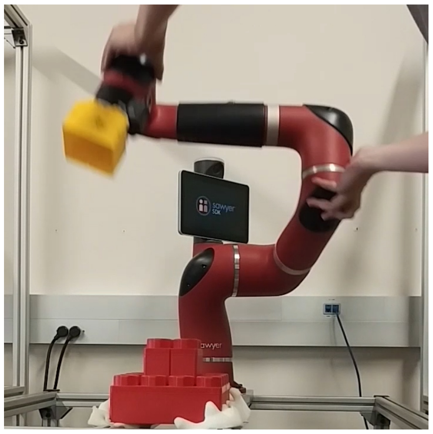


Figure 3.1: We trained a Sawyer robot to stack Lego blocks together using maximum entropy reinforcement learning algorithm called soft Q-learning. Training a policy from scratch takes less than two hours, and the learned policy is extremely robust against perturbations.

general EBM, becomes intractable. To that end, we borrow from the recent literature on EBMs to devise an approximate sampling procedure based on training a separate sampling network, which is optimized to produce unbiased samples from the policy EBM.

The principal contribution of this chapter is a tractable, efficient algorithm for optimizing arbitrary multimodal stochastic policies represented by energy-based models. In our experimental evaluation, we explore three potential applications of our approach. First, we demonstrate improved exploration performance in tasks with multimodal objectives, where conventional deterministic or unimodal methods are at high risk of falling into suboptimal local optima. Second, we explore how our method can be used to transfer skill from underdefined tasks to more specific tasks by showing that stochastic energy-based policies can serve as a much better initialization for learning new skills than either random policies or policies pretrained with conventional maximum reward objectives. Third, we demonstrate that soft Q-learning can be used to acquire real-world robot skills more efficiently than the prior methods, and that the resulting policies are extremely robust to perturbations (Figure 3.1).

3.2 Soft Value Functions and Energy Based-Models

In this section, we present a few useful identities that we will build on in our algorithm. Optimizing the maximum entropy objective in Equation 1.1 (more precisely, the discounted infinite horizon objective in Equation A.1) provides us with a framework for training stochastic policies, but we must still choose a representation for these policies. The choices in prior work include discrete multinomial distributions (O'Donoghue et al., 2016) and Gaussian distributions (Rawlik et al., 2012). However, if we want to use a very general class of distributions that can represent complex, multimodal behaviors, we can instead opt for using general energy-based policies of the form

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp\left(-\mathcal{E}(\mathbf{s}_t, \mathbf{a}_t)\right), \tag{3.1}$$

where $\mathcal{E}$ is an energy function that could be represented, for example, by a deep neural network. If we use a universal function approximator for $\mathcal{E}$, we can represent any distribution

$\pi(\mathbf{a}_t|\mathbf{s}_t)$. There is a close connection between such energy-based models and soft versions of value functions and Q-functions, where we set $\mathcal{E}(\mathbf{s}_t, \mathbf{a}_t) = -\frac{1}{\alpha}Q(\mathbf{s}_t, \mathbf{a}_t)$ and use the following lemma:

Lemma 1. *Let the soft Q-function be defined by*

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\tau\sim\rho_{\pi^*}}\left[\sum_{l=1}^{\infty}\gamma^l\left(r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha\log\left(\pi^*(\mathbf{a}_{t+l}|\mathbf{s}_{t+l})\right)\right)\middle|\mathbf{s}_t, \mathbf{a}_t\right], \quad (3.2)$$

and the soft value function by

$$V^*(\mathbf{s}_t) = \alpha\log\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q^*(\mathbf{s}_t, \mathbf{a}')\right)d\mathbf{a}'. \quad (3.3)$$

Then the optimal infinite horizon, maximum entropy policy is given by

$$\pi^*(\mathbf{a}_t|\mathbf{s}_t) = \exp\left(\frac{1}{\alpha}\left(Q^*(\mathbf{s}_t, \mathbf{a}_t) - V^*(\mathbf{s}_t)\right)\right). \quad (3.4)$$

Proof. See Appendix B.1 as well as (Ziebart, 2010). □

Lemma 1 connects the maximum entropy objective in and energy-based models, where $\frac{1}{\alpha}Q(\mathbf{s}_t, \mathbf{a}_t)$ acts as the negative energy, and $\frac{1}{\alpha}V(\mathbf{s}_t)$ serves as the log-partition function. As with the standard Q-function and value function, we can relate the soft Q-function to the soft value function at a future state via the soft Bellman equation:

Lemma 2. *The soft Q-function in Equation 3.2 satisfies the soft Bellman equation*

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V^*(\mathbf{s}_{t+1})\right], \quad (3.5)$$

where the soft value function V^* is given by Equation 3.3.

Proof. See Appendix B.1, as well as (Ziebart, 2010). □

The soft Bellman equation is a generalization of the conventional (hard) equation, and we can recover the more standard equation as $\alpha \to 0$, which causes Equation 3.3 to approach the hard maximum over the actions. In the next section, we discuss how we can use these identities to derive a Q-learning style algorithm for learning maximum entropy policies, and how we can make this practical for arbitrary soft Q-function representations via an approximate inference procedure.

3.3 Training Expressive Energy-Based Models via Soft Q-Learning

In this section, we present our proposed reinforcement learning algorithm, which is based on the soft Q-function described in the previous section, but can be implemented via a tractable stochastic gradient descent procedure with approximate sampling. We first describe the general case of soft Q-learning, and then present the inference procedure that makes it tractable to use with deep neural network representations in high-dimensional continuous state and action spaces.

Soft Q-Iteration

We can obtain a solution to Equation 3.5 by iteratively updating estimates of V^* and Q^*. This leads to a fixed-point iteration that resembles Q-iteration:

Theorem 1. *Soft Q-iteration. Let Q and V be bounded and $\int_{\mathcal{A}} \exp(\frac{1}{\alpha}Q(\mathbf{s}_t, \mathbf{a}'))d\mathbf{a}' < \infty$, $\forall \mathbf{s}_t \in \mathcal{S}$, and assume that $Q^* < \infty$ exists. Then the fixed-point iteration*

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V(\mathbf{s}_{t+1})\right],\ \forall \mathbf{s}_t, \mathbf{a}_t \tag{3.6}$$

$$V(\mathbf{s}_t) \leftarrow \alpha \log\left(\int_{\mathcal{A}} \exp\left(\frac{1}{\alpha}Q(\mathbf{s}_t, \mathbf{a}')\right) d\mathbf{a}'\right),\ \forall \mathbf{s}_t \tag{3.7}$$

converges to Q^* and V^*, respectively.

Proof. See Appendix B.1 as well as (Fox et al., 2016). □

We refer to the updates in Equation 3.6 and Equation 3.7 as the soft Bellman backup operation that acts on the soft value function, and denote it by $\mathcal{T}$. The maximum entropy policy in Equation 3.4 can be recovered by iteratively applying this operator until convergence. However, there are several practicalities that need to be considered in order to make use of the algorithm. First, the soft Bellman backup cannot be performed exactly in continuous or large state and action spaces, and second, sampling from the energy-based model in Equation 3.4 is intractable in general. We address these challenges next.

Soft Q-Learning

This section discusses how the soft Bellman backup in Theorem 1 can be implemented in a practical algorithm that uses a finite set of samples from the environment, resulting in a method similar to Q-learning. We can show that the soft Bellman backup is a contraction (see Appendix B.1), and thus the optimal value function is the fixed point of the soft Bellman backup, and we can find it by optimizing for a soft Q-function for which the soft Bellman error $|\mathcal{T}Q - Q|$ is minimized at all states and actions. While this procedure is still intractable due to the integral in Equation 3.7 and the infinite set of all states and actions, we can express it as a stochastic optimization problem, which leads to a stochastic gradient descent update procedure. We model the soft Q-function with a function approximator with parameters θ and denote it as $Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$.

To convert Theorem 1 into a stochastic optimization problem, we first express the soft value function in terms of an expectation via importance sampling:

$$V_\theta(\mathbf{s}_t) = \alpha \log \mathbb{E}_{q_{\mathbf{a}'}} \left[\frac{\exp\left(\frac{1}{\alpha} Q_\theta(\mathbf{s}_t, \mathbf{a}')\right)}{q_{\mathbf{a}'}(\mathbf{a}')} \right], \tag{3.8}$$

where $q_{\mathbf{a}'}$ is an arbitrary, non-zero distribution over the action space. Second, by noting the identity $g_1(x) = g_2(x) \ \forall x \in \mathbb{X} \ \Leftrightarrow \ \mathbb{E}_{x\sim q}\left[(g_1(x) - g_2(x))^2\right] = 0$, where q is any strictly positive density function on $\mathbb{X}$, we can express soft Q-iteration in an equivalent form of the minimization of

$$J_Q(\theta) = \mathbb{E}_{\mathbf{s}_t\sim q_{\mathbf{s}}, \mathbf{a}_t\sim q_{\mathbf{a}}} \left[\frac{1}{2} \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \left(r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\theta}}(\mathbf{s}_{t+1}) \right] \right) \right)^2 \right], \tag{3.9}$$

where $q_{\mathbf{s}}, q_{\mathbf{a}}$ are positive densities over $\mathcal{S}$ and $\mathcal{A}$, respectively, and $V_{\bar{\theta}}(\mathbf{s}_{t+1})$ is a target value function given by Equation 3.8 and θ being replaced by the target parameters $\bar{\theta}$. This stochastic optimization problem can be solved approximately using stochastic gradient descent and tuples $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ sampled from a pool of observed transitions. An ideal choice for For $q_{\mathbf{a}'}$ would be the current policy, which would produce an unbiased estimate of the soft value as can be confirmed by substitution. However, our approach is to use an implicit density model for the policy, which does not allow computing the action probabilities required in Equation 3.8, and thus we choose to use uniform samples instead. Even though uniform sampling does not scale well to higher dimensional spaces, actions are typically relatively low-dimensional, and we found uniform sampling work well in practice. This overall procedure yields an iterative approach that optimizes over the Q-values. We also need a way to sample from the policy $\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp\left(\frac{1}{\alpha} Q_\theta(\mathbf{s}_t, \mathbf{a}_t)\right)$. Since the form of the policy is so general, sampling from it is intractable. We will therefore use an approximate sampling procedure, as discussed in the following section.

Approximate Sampling and Stein Variational Gradient Descent

Existing approaches that sample from energy-based distributions generally fall into one of two categories: methods that use Markov chain Monte Carlo (MCMC) based sampling (Sallans & Hinton, 2004), and methods that learn a stochastic sampling network trained to output approximate samples from the target distribution (Zhao et al., 2016; Kim & Bengio, 2016). Since sampling via MCMC is not tractable when the inference must be performed online (e.g. when executing a policy), we will use a sampling network based on Stein variational gradient descent (SVGD) (Liu & Wang, 2016) and amortized SVGD (Wang & Liu, 2016). Amortized SVGD provides us with a stochastic sampling network that we can query for fast sample generation, and it can be shown to converge to an accurate estimate of the posterior distribution of an EBM.

Formally, we want to learn a state-conditioned stochastic neural network $\mathbf{a}_t = f_\phi(\xi; \mathbf{s}_t)$, parameterized by ϕ, that maps noise samples ξ drawn from a spherical Gaussian, or other arbitrary distribution, into unbiased action samples from the target EBM corresponding to Q_θ. We denote the induced distribution of the actions as $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$, and we want to find parameters ϕ so that the induced distribution approximates the energy-based distribution in terms of the KL divergence

$$J_\pi(\phi; \mathbf{s}_t) = \mathrm{D}_{\mathrm{KL}}\left(\pi_\phi(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \exp\left(\frac{1}{\alpha}\left(Q_\theta(\mathbf{s}_t, \,\cdot\,) - V_\theta(\mathbf{s}_t)\right)\right)\right). \tag{3.10}$$

Suppose we have a set of independent, state conditioned action samples $\{\mathbf{a}_t^{(i)} = f_\phi(\xi^{(i)}; \mathbf{s}_t)\}$, given by the sampling network, and we "perturb" them in some directions $\Delta f_\phi(\xi^{(i)}; \mathbf{s}_t)$, then the KL divergence induced by the samples can be reduced. Stein variational gradient descent (Liu & Wang, 2016) provides the most greedy directions as a functional

$$\Delta f_\phi(\,\cdot\,; \mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi}\left[\kappa(\mathbf{a}_t, f_\phi(\,\cdot\,; \mathbf{s}_t)) \nabla_{\mathbf{a}'} Q_\theta(\mathbf{s}_t, \mathbf{a}')\big|_{\mathbf{a}'=\mathbf{a}_t} + \alpha\, \nabla_{\mathbf{a}'} \kappa(\mathbf{a}', f_\phi(\,\cdot\,; \mathbf{s}_t))\big|_{\mathbf{a}'=\mathbf{a}_t}\right], \tag{3.11}$$

where κ is a kernel function (typically a Gaussian, see details in Appendix C.1). To be precise, Δf^ϕ is the optimal direction in the reproducing kernel Hilbert space of κ, and is thus not strictly speaking the gradient of Equation 3.10, but it turns out that we can set $\frac{\partial J_\pi}{\partial \mathbf{a}_t} \propto \Delta f_\phi$ as explained in (Wang & Liu, 2016). With this assumption, we can use the chain rule and backpropagate the Stein variational gradient into the policy network according to

$$\frac{\partial J_\pi(\phi; \mathbf{s}_t)}{\partial \phi} \propto \mathbb{E}_\xi\left[\Delta f_\phi(\xi; \mathbf{s}_t) \frac{\partial f_\phi(\xi; \mathbf{s}_t)}{\partial \phi}\right], \tag{3.12}$$

and use any gradient-based optimization method to learn the optimal sampling network parameters.

Algorithm Summary

To summarize, we propose the soft Q-learning algorithm for learning maximum entropy policies in continuous domains (Algorithm 1). The algorithm proceeds by alternating between collecting new experience from the environment and updating the soft Q-function and sampling network parameters. The experience is stored in a replay memory buffer $\mathcal{D}$ as standard in deep Q-learning (Mnih et al., 2013), and the parameters are updated using random minibatches from this memory. The soft Q-function updates use a delayed version of the target values (Mnih et al., 2015). For optimization, we use the ADAM (Kingma & Ba, 2015) optimizer and empirical estimates of the gradients, which we denote by $\hat{\nabla}$. We have additionally included an exact formula for the empirical gradient of the policy parameters and other implementation details in Appendix C.1.

Algorithm 1 Soft Q-learning

```
Input: $\theta, \phi$                                          ▷ Initial parameters
  $\bar{\theta} \leftarrow \theta$, $\bar{\phi} \leftarrow \phi$                       ▷ Assign target parameters
  $\mathcal{D} \leftarrow \emptyset$                                      ▷ Initialize empty replay buffer
  for each epoch do
      for each $t$ do
          Collect experience
          $\mathbf{a}_t \leftarrow f^{\phi}(\xi; \mathbf{s}_t)$ where $\xi \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$        ▷ Sample an action for $\mathbf{s}_t$ using $f^{\phi}$
          $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$              ▷ Sample next state from the environment
          $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$    ▷ Save the transition in the replay memory
          Update soft Q-function parameters
          $\{(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}, r_t^{(i)}, \mathbf{s}_{t+1}^{(i)})\}_{i=0}^{N} \sim \mathcal{D}$       ▷ Sample a minibatch from the replay memory
          $\{\mathbf{a}^{(i,j)}\}_{j=0}^{M} \sim q_{\mathbf{a}'}$                    ▷ Sample $M$ uniform actions for each $\mathbf{s}_{t+1}^{(i)}$
          $v^{(i)} \leftarrow \hat{V}_{\bar{\theta}}(\mathbf{s}_{t+1}^{(i)})$              ▷ Compute empirical soft values with Equation 3.8
          $\mathbf{g}_{\theta}^{(i)} \leftarrow \hat{\nabla}_{\theta} J_Q$                  ▷ Compute empirical gradient of Equation 3.9
          $\theta \leftarrow \mathrm{ADAM}(\theta, \mathbf{g}_{\theta}^{(i)})$          ▷ Update $\theta$ using ADAM and stochastic gradients
          Update policy
          $\{\xi^{(i,j)}\}_{j=0}^{M} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$                   ▷ Sample $M$ latents for each state
          $\mathbf{a}_t^{(i,j)} \leftarrow f_{\phi}(\xi^{(i,j)}, \mathbf{s}_t^{(i)})$                             ▷ Evaluate actions
          $\delta\mathbf{a}_t^{(i,j)} \leftarrow \Delta f_{\phi}$  ▷ Evaluate empirical Stein variational gradient using Equation 3.11
          $\mathbf{g}_{\phi}^{(i)} \leftarrow \hat{\nabla}_{\phi} J_{\pi}$               ▷ Compute empirical estimate of Equation 3.12
          $\phi \leftarrow \mathrm{ADAM}(\phi, \mathbf{g}_{\phi}^{(i)})$                ▷ Update policy parameters using ADAM
      end for
      if epoch $mod$ update_interval = 0 then
          $\bar{\theta} \leftarrow \theta$, $\bar{\phi} \leftarrow \phi$                       ▷ Update target parameters
      end if
  end for
Output: $\theta$, $\phi$                                      ▷ Optimized parameters
```

3.4 Experiments

Our experiments aim to answer the following questions: (1) Does our soft Q-learning algorithm accurately capture a multimodal policy distribution? (2) Can soft Q-learning with energy-based policies aid exploration for complex tasks that require tracking multiple modes? (3) Can a maximum entropy policy serve as a good initialization for fine-tuning on different tasks, when compared to pretraining with a standard deterministic objective? (4) Can soft Q-learning be used to learn policies on real-world robotic platforms. We compare our algorithm to DDPG (Lillicrap et al., 2015), which has been shown to achieve better sample efficiency on the continuous control problems than other recent techniques such as REIN-

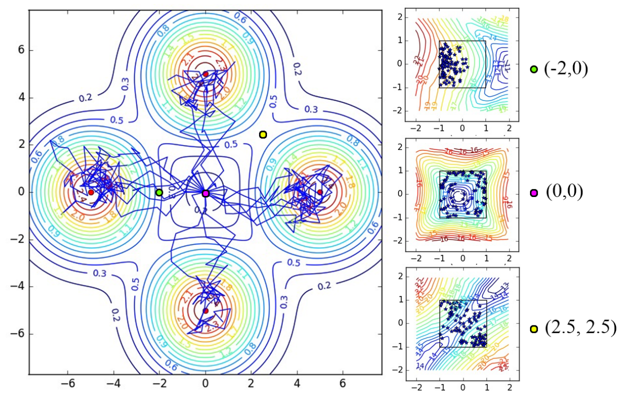


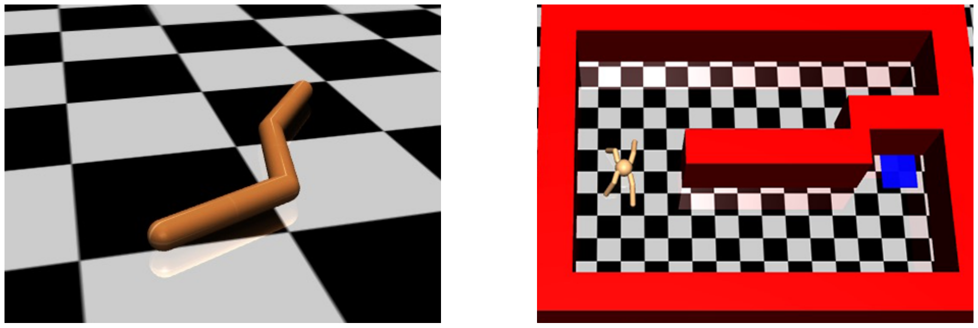
Figure 3.2: Illustration of the 2D multi-goal environment. Left: trajectories from a policy learned with our method (solid blue lines). The x and y axes correspond to 2D positions (states). The agent is initialized at the origin. The goals are depicted as red dots, and the level curves show the reward. Right: Q-values at three selected states, depicted by level curves (red: high values, blue: low values). The x and y axes correspond to 2D velocity (actions), which is bounded between ± 1. Actions sampled from the policy are shown as blue stars. Note that, in regions (e.g. $(2.5, 2.5)$) between the goals, the method chooses multimodal actions.

FORCE (Williams, 1992), TRPO (Schulman et al., 2015a), and A3C (Mnih et al., 2016), which are policy gradient methods and cannot thus make use of off-policy data. The detailed experimental setup can be found in Appendix C.1. Videos of the simulated experiments[1], real-world experiments[2], and example source code[3] are available online.

Didactic Example: Multi-Goal Environment

In order to verify that amortized SVGD can correctly draw samples from energy-based policies of the form $\exp(Q_\theta(\mathbf{s}_t, \mathbf{a}_t))$, and that our complete algorithm can successful learn to represent multimodal behavior, we designed a simple "multi-goal" environment, in which the agent is a 2D point trying to reach one of four symmetrically placed goals. The reward is defined as a mixture of Gaussians, with means placed at the goal positions. An optimal strategy is to go to an arbitrary goal, and the optimal maximum entropy policy should be able to choose each of the four goals at random. The final policy obtained with our method is illustrated in Figure 3.2. The soft Q-function (the small figures on the right) is a non-trivial function of the state and action, being unimodal at $\mathbf{s} = (-2, 0)$, convex at $\mathbf{s} = (0, 0)$, and bimodal at $\mathbf{s} = (2.5, 2.5)$. The stochastic policy samples actions closely following the energy landscape, hence representing diverse trajectories that lead to all four goals. In comparison, a policy trained with DDPG randomly commits to a single goal.

[1]https://sites.google.com/view/softqlearning/home/
[2]https://sites.google.com/view/composing-real-world-policies/
[3]https://github.com/haarnoja/softqlearning/



(a) Swimming snake (b) Quadrupedal robot

Figure 3.3: Simulated robots used in our experiments.

Learning Multimodal Policies for Exploration

Though not all environments have a clear multimodal reward landscape as in the multi-goal example, multimodality is prevalent in a variety of tasks. For example, a chess player might try various strategies before settling on one that seems most effective, and an agent navigating a maze may need to try various paths before finding the exit. During the learning process, it is often best to keep trying multiple available options until the agent is confident that one of them is the best (similar to a bandit problem (Lai & Robbins, 1985)). However, deep RL algorithms for continuous control typically use unimodal action distributions, which are not well suited to capture such multimodality. As a consequence, such algorithms may prematurely commit to one mode and converge to suboptimal behavior.

To evaluate how maximum entropy policies might aid exploration, we constructed simulated continuous control environments where tracking multiple modes is important for success. The first experiment uses a simulated swimming snake shown in Figure 3.3a, which receives a reward equal to its speed along the x-axis, either forward or backward. However, once the swimmer swims far enough forward, it crosses a “finish line” and receives a larger reward. Therefore, the best learning strategy is to explore in both directions until the bonus reward is discovered, and then commit to swimming forward. As illustrated in Figure 3.4 and Figure 3.5a, our method is able to recover this strategy, keeping track of both modes until the finish line is discovered. All stochastic policies eventually commit to swimming forward. On the other hand, DDPG commits to an arbitrary mode prematurely, with only 80% of the policies converging on a forward motion and 20% choosing the suboptimal backward mode.

The second experiment studies a more complex task with a continuous range of equally good options prior to discovery of a sparse reward goal. In this task, a quadrupedal 3D robot (adapted from Schulman et al. (2015b)) needs to find a path through a maze to a target position (Figure 3.3). The reward function is a Gaussian centered at the target. The agent may choose either the upper or lower passage, which appear identical at first, but the upper passage is blocked by a barrier. Similarly to the swimmer experiment, the optimal strategy requires exploring both directions and choosing the better one. Figure 3.5b compares the

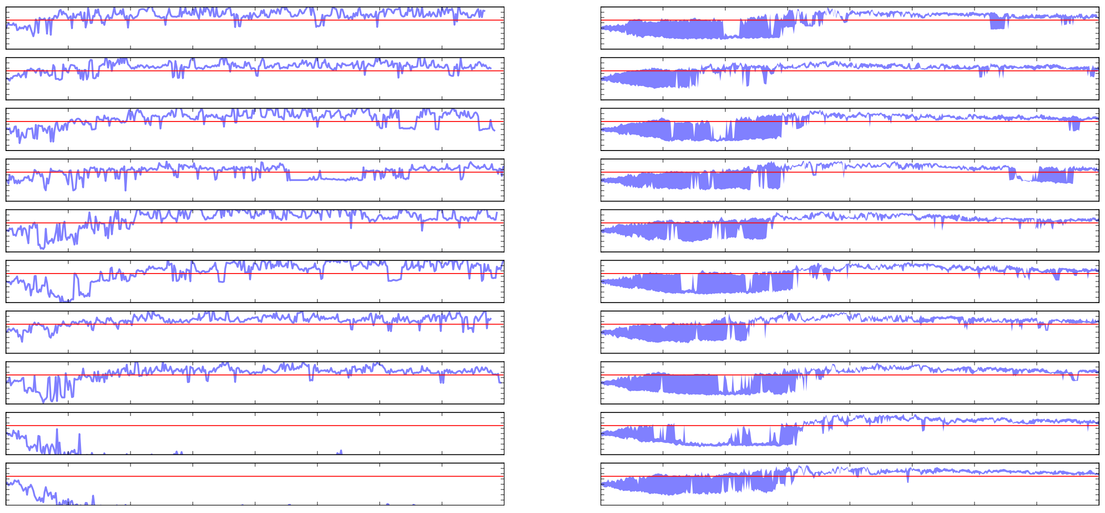
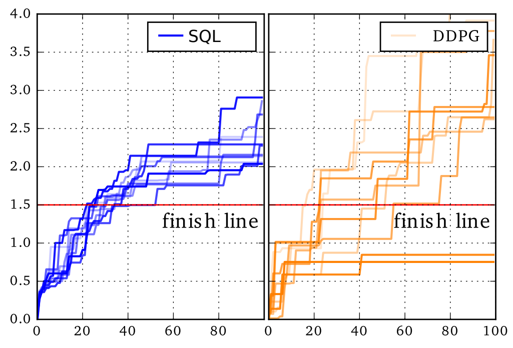

(a) DDPG (b) Soft Q-Learning

Figure 3.4: Forward swimming distance achieved by each policy. Each row is a policy with a unique random seed. Horizontal axis corresponds to the training iteration, and vertical axis to the x-coordinate of the agent's center of mass. When the agent passes the "finish line" shown in red, it receives a large positive reward. The blue shaded region shows the maximum and minimum distance travelled of the evaluation rollouts, and they are all equal for DDPG policy, which is deterministic, and different for soft Q-learning policy, which is stochastic. Our method is able to explore both directions before the policy commits to the better one.

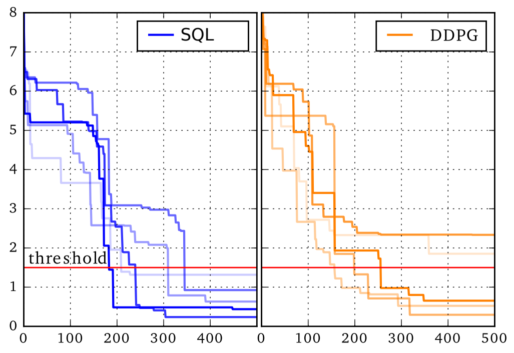
performance of DDPG and our method. The curves show the minimum distance to the target achieved so far and the threshold equals the minimum possible distance if the robot chooses the upper passage. Therefore, successful exploration means reaching below the threshold. All policies trained with our method manage to succeed, while only 60% policies trained with DDPG converge to choosing the lower passage.

Accelerating Training on Complex Tasks with Pretrained Maximum Entropy Policies

A standard way to accelerate deep neural network training is task-specific initialization (Goodfellow et al., 2016), where a network trained for one task is used as initialization for another task. The first task might be something highly general, such as classifying a large image dataset, while the second task might be more specific, such as fine-grained classification with a small dataset. Pretraining has also been explored in the context of RL (Shelhamer et al., 2016). However in RL, optimal policies are often deterministic, which makes them poor initializers for new tasks. In this section, we explore how our energy-based policies can be trained with fairly broad objectives to produce a good initializer for learning more



(a) Swimmer (higher is better)



(b) Quadruped (lower is better)

Figure 3.5: Comparison of soft Q-learning (SQL) and DDPG on the swimmer snake task and the quadrupedal robot maze task. (a) Shows the maximum traveled forward distance since the beginning of training for several runs of each algorithm; there is a large reward after crossing the finish line. (b) Shows our method was able to reach a low distance to the goal faster and more consistently. The different lines show the minimum distance to the goal since the beginning of training. For both domains, all runs of our method cross the threshold line, acquiring the more optimal strategy, while some runs of DDPG do not.

specific tasks.

We demonstrate this on a variant of the quadrupedal robot task. The pretraining phase involves learning to locomote in an arbitrary direction, with a reward that simply equals the speed of the center of mass. The resulting policy moves the agent quickly to an random direction as shown in Figure 3.6. This pretraining is similar in some ways to recent work on modulated controllers (Heess et al., 2016) and hierarchical models (Florensa et al., 2017). However, in contrast to these prior works, we do not require any task-specific high-level goal generator or reward. Figure 3.7 shows a variety of test environments that we used to fine-tune the pretrained policy for a specific task. In the hallway environments, the reward function is the same as in the pretraining phase, but the walls block sideways motion, so the optimal solution is to run in a particular direction. Narrow hallways require choosing a more specific direction, but also allow the agent to use the walls to funnel itself forward. The U-shaped maze requires the agent to learn a curved trajectory in order to arrive at the target, with the reward given by a Gaussian bump at the target location.

The pretrained policy explores the space extensively and in all directions. This gives a good initialization for the policy, allowing it to learn the behaviors in the test environments more quickly than training a policy with DDPG from a random initialization, as shown in Figure 3.8. We also evaluated an alternative pretraining method based on deterministic policies learned with DDPG. However, deterministic pretraining chooses an arbitrary but consistent direction in the training environment, providing a poor initialization for fine-tuning to a specific task.

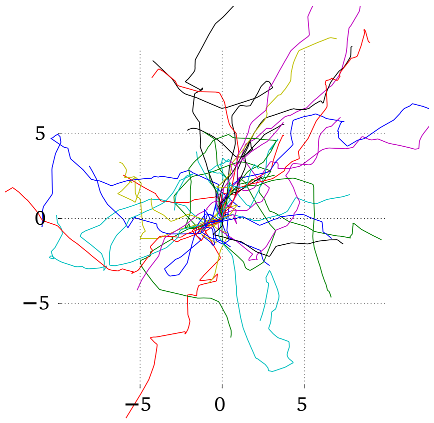


Figure 3.6: The figure shows rollouts of a policy that is pretrained to maximize the speed of a quadrupedal robot. Each path corresponds to one trajectory from the single stochastic policy trained with soft Q-learning. Figure D.1 in Appendix D.1 shows more examples of policies trained with varying temperature coefficients α.

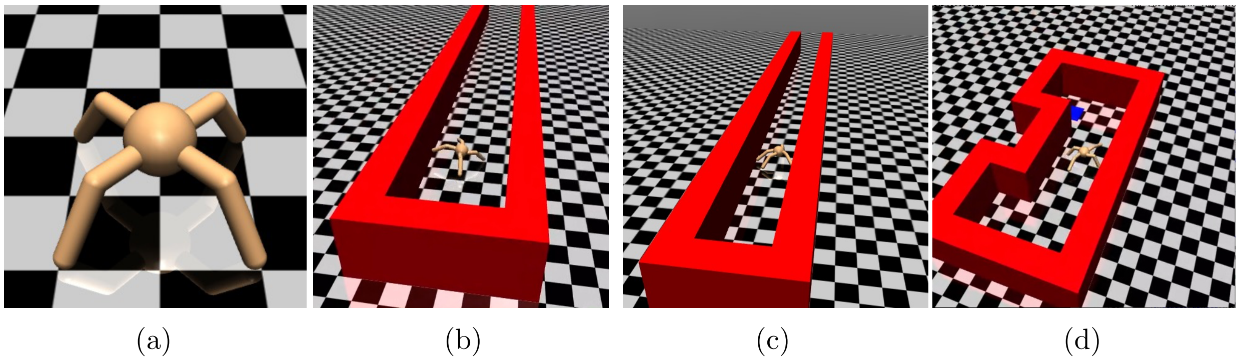


Figure 3.7: Quadrupedal robot (a) was trained to walk in random directions in an empty pretraining environment and then fine-tuned on a variety of tasks, including a wide (b), narrow (c), and U-shaped hallway (d).

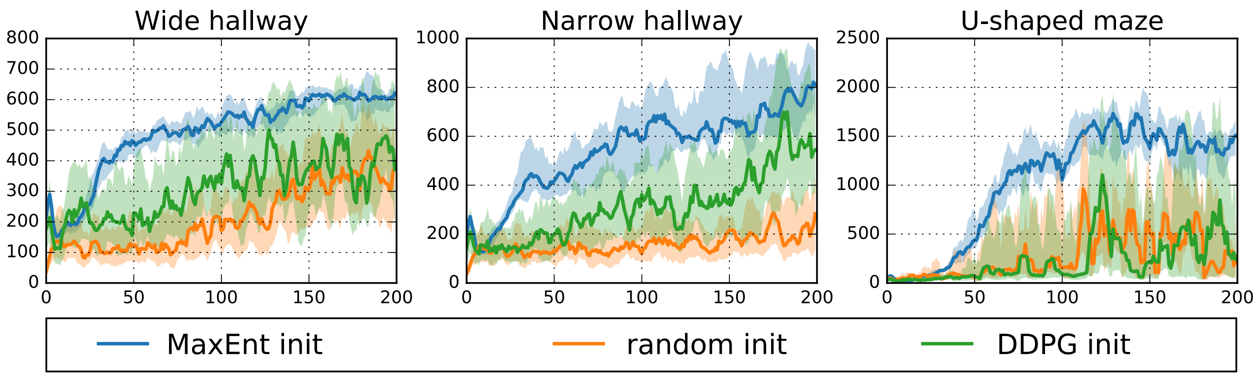


Figure 3.8: Performance in the downstream task with fine-tuning (MaxEnt) or training from scratch (DDPG). The x-axis corresponds to the training iteration and the y-axis to the average discounted return. Solid lines show the average values over 10 random seeds, and the shaded regions correspond to one standard deviation of the return over the seeds.

Soft Q-Learning in the Real-World

We also trained policies on a real-world robotic manipulator, and our results indicate soft Q-learning can acquire skills quickly and reliably and yields better sample complexity over existing model-free deep RL methods.[4] We evaluate our method on Cartesian reaching and Lego block stacking on a real-world Sawyer robot with 7 degrees of freedom. The actions are the torque commands at each joint, and the observations are the joint angles and angular velocities as well as the end-effector position. For the Lego stacking task, we also include downward force estimated from the actuator currents as observations.

We compare soft Q-learning to deep deterministic policy gradients, and normalized advantage functions (NAF) in terms of sample complexity and final error, and a comparison is shown in Figure 3.9. Each experiment was repeated three times to analyze the variability of the methods. Since SQL does not rely on external exploration, we simply show training performance, whereas for DDPG and NAF, we show test performance at regular intervals, with the exploration noise switched off to provide a fair comparison. Soft Q-learning (blue curve) solves the task in about 10 minutes, whereas DDPG (green) and NAF (red) are substantially slower and exhibit large variation between training runs. We also trained a policy with SQL to reach randomly selected points (orange) that are provided as input to the policy, with a different target point selected for each episode. Learning this policy was almost as fast as training for a fixed target, but because some targets are easier to reach than others, the policy exhibited larger variation in terms of the final error of the end-effector position.

In the final experiment, we used SQL to train a policy for stacking Lego blocks to test its ability to exercise precise control in the presence of contact dynamics (Figure 3.10). The goal is to position the end effector, which holds a Lego block, into a position and orientation that successfully stacks it on top of another block. To reduce variance in the end-effectors pose, we augmented the reward function with an additional negative log-distance term that

[4]This experiment was originally presented in (Haarnoja et al., 2018b).

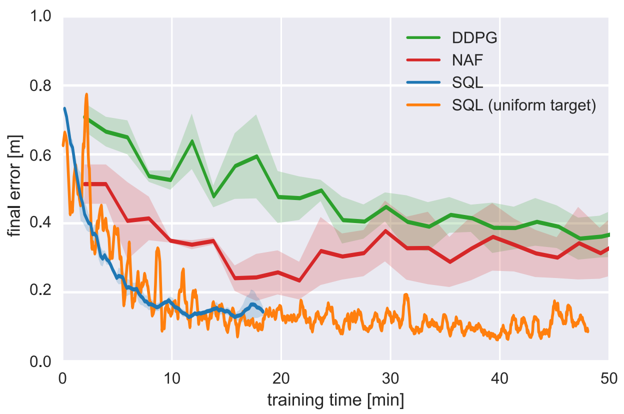


Figure 3.9: The learning curve of DDPG (green), NAF (red), and SQL (blue) on the Sawyer robot when trained to move its end effector to a specific location. Soft Q-learning converged much faster than the other methods. We also trained SQL to reach randomly sampled end-effector locations by concatenating the desired location to the observation vector (orange). Soft Q-learning solves this task just as quickly. Soft Q-learning curves show the moving average over 10 training episodes.

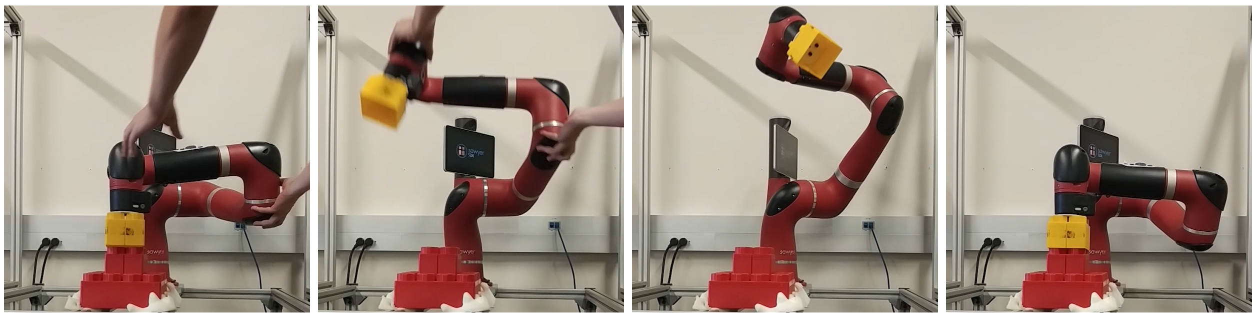


Figure 3.10: A Lego stacking policy can be learned in less than two hours. The learned policy is remarkably robust against perturbations: the robot is able to recover and successfully stack the Lego blocks together after being pushed into a state that is substantially different from typical trajectories.

incentivizes higher accuracy when the block is nearly inserted, and we also included a reward for downward force to overcome the friction forces. After half an hour of training, the robot was able to successfully insert the Lego block for the first time, and in two hours, the policy was fully converged. We evaluated the task visually by inspecting if all of the four studs were in contact and observed a success rate of 100 % on 20 trials of the final policy. The resulting policy does not only solve the task consistently, but it is also surprisingly robust to disturbances. To test robustness, we forced the arm into configurations that are far from the states that it encounters during normal execution, and the policy was able to recover every time and eventually solve the task as we illustrate in Figure 3.10. We believe the robustness can be attributed to the effective and diverse exploration that is natural to maximum entropy policies, and, to our knowledge, no other prior deep RL work has shown similar results.

3.5 Conclusion

We presented a method for learning stochastic energy-based policies with approximate inference via Stein variational gradient descent (SVGD). Our approach can be viewed as a type of soft Q-learning method, with the additional contribution of using approximate inference to obtain complex multimodal policies. Our experimental results show that our method can effectively capture complex multimodal behavior on problems ranging from toy point tasks to complex torque control of simulated walking and swimming robots, and can learn real-world robotic manipulation substantially faster than prior methods, NAF and DDPG. The applications of training such stochastic policies include improved exploration in the case of multimodal objectives and transfer learning, which involves pretraining general-purpose stochastic policies and efficiently fine-tuning the for a more specific task.

Chapter 4

Soft Actor-Critic

In the previous chapter, we proposed a reinforcement learning algorithm based on soft Q-learning that solves directly for the optimal soft Q-function, from which the optimal policy can be recovered. In this chapter, we will discuss how we can devise a soft actor-critic algorithm through a soft policy iteration formulation, where we instead evaluate the soft Q-function of the current policy and update the policy through an off-policy gradient update. Though such algorithms have previously been proposed for conventional reinforcement learning, our method is, to our knowledge, the first off-policy actor-critic method in the maximum entropy reinforcement learning framework. The method achieves state-of-the-art performance on a range of continuous control benchmark tasks, outperforming prior on-policy and off-policy methods. Furthermore, we demonstrate that, in contrast to other off-policy algorithms, our approach is very stable, achieving very similar performance across different random seeds.

4.1 Overview

Maximum entropy formulation can provide a substantial improvement in exploration and robustness, yields policies that are robust in the face of model and estimation errors (Ziebart, 2010), and as we demonstrated in the previous chapter, improves exploration by acquiring diverse behaviors. Prior work has proposed model-free deep RL algorithms that perform on-policy learning with entropy maximization (O'Donoghue et al., 2016), as well as off-policy methods based on soft Q-learning and its variants (Schulman et al., 2017a; Nachum et al., 2017; Haarnoja et al., 2017). However, the on-policy variants suffer from poor sample complexity (Section 1.1), while the off-policy variants require complex approximate inference procedures in continuous action spaces (Chapter 3).

In this chapter, we demonstrate that we can devise a simple off-policy maximum entropy actor-critic algorithm, which we call soft actor-critic (SAC), which provides for both sample-efficient learning and stability. This algorithm extends readily to very complex, high-dimensional tasks, such as the Humanoid benchmark (Duan et al., 2016) with 21 action dimensions, where off-policy methods such as DDPG typically struggle to obtain good results (Gu et al., 2017). Soft actor-critic also avoids the complexity and potential instability associated with approximate inference in prior off-policy maximum entropy algorithms based on soft Q-learning. We present a convergence proof for policy iteration in the maximum entropy framework, and then introduce a new algorithm based on an approximation to this procedure that can be practically implemented with deep neural networks, which we call soft actor-critic. We present empirical results that show that soft actor-critic attains a substantial improvement in both performance and sample efficiency over both off-policy and on-policy prior methods. We also compare to twin delayed deep deterministic (TD3) policy gradient algorithm (Fujimoto et al., 2018), which is a recent method that proposes a deterministic algorithm that substantially improves on DDPG.

4.2 From Soft Policy Iteration to Soft Actor-Critic

Our off-policy soft actor-critic algorithm can be derived starting from a maximum entropy variant of the policy iteration method. We first present this derivation, verify that the corresponding algorithm converges to the optimal policy from its density class, and then present a practical deep reinforcement learning algorithm based on this theory.

Soft Policy Iteration

We begin by deriving soft policy iteration, a general algorithm for learning optimal maximum entropy policies that alternates between *soft policy evaluation* and *soft policy improvement* in the maximum entropy framework. Our derivation is based on a tabular setting, to enable theoretical analysis and convergence guarantees, and we extend this method into the general continuous setting later. We show that soft policy iteration converges to the optimal policy within a set of policies which might correspond, for instance, to a set of parameterized Gaussian distributions.

In the soft policy evaluation step, we wish to compute the value of a policy π according to the maximum entropy objective. For a fixed policy, the soft Q-value can be computed iteratively, starting from any function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and repeatedly applying a modified Bellman backup operator $\mathcal{T}^\pi$ given by

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V(\mathbf{s}_{t+1}) \right], \tag{4.1}$$

where

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi} \left[Q(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi(\mathbf{a}_t | \mathbf{s}_t) \right] \tag{4.2}$$

is the soft state value function. We can obtain the soft value function for any policy π by repeatedly applying $\mathcal{T}^\pi$ as formalized below.

Lemma 3 (Soft Policy Evaluation). *Consider the soft Bellman backup operator $\mathcal{T}^\pi$ in Equation 4.1 and a mapping $Q^0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with $|\mathcal{A}| < \infty$, and define $Q^{k+1} = \mathcal{T}^\pi Q^k$. Then the sequence Q^k will converge to the soft Q-function of π as $k \to \infty$.*

Proof. See Appendix B.3. □

In the policy improvement step, we update the policy towards the exponential of the new soft Q-function. This particular choice of update can be guaranteed to result in an improved policy in terms of its soft value. Since in practice, we prefer policies that are tractable, we will additionally restrict the policy to some set of policies Π, which can correspond, for example, to a parameterized family of distributions such as Gaussians. To account for the constraint $\pi \in \Pi$, we project the improved policy into the desired set of policies. While in principle we could choose any projection, it will turn out to be convenient to use the information projection defined in terms of the Kullback-Leibler divergence. In the other words, in the policy improvement step, for each state, we update the policy according to

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}} \left(\pi'(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(\frac{1}{\alpha} Q^{\pi_{\text{old}}}(\mathbf{s}_t, \,\cdot\,)\right)}{Z^{\pi_{\text{old}}}(\mathbf{s}_t)} \right). \tag{4.3}$$

The partition function $Z^{\pi_{\text{old}}}(\mathbf{s}_t)$ normalizes the distribution, and while it is intractable in general, it does not contribute to the gradient with respect to the new policy and can thus be ignored, as we note in the next section. For this projection, we can show that the new, projected policy has a higher value than the old policy with respect to the maximum entropy objective. We formalize this result in Lemma 4.

Lemma 4 (Soft Policy Improvement). *Let $\pi_{\text{old}} \in \Pi$ and let π_{new} be the optimizer of the minimization problem defined in Equation 4.3. Then $Q^{\pi_{\text{new}}}(\mathbf{s}_t, \mathbf{a}_t) \geq Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t)$ for all $(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{S} \times \mathcal{A}$ with $|\mathcal{A}| < \infty$.*

Proof. See Appendix B.3. □

The complete soft policy iteration algorithm alternates between soft policy evaluation and soft policy improvement, and it will provably converge to the optimal maximum entropy policy among the policies in Π (Theorem 2). Although this algorithm will provably find the optimal solution, we can perform it in its exact form only in the tabular case. Therefore, we will next approximate the algorithm for continuous domains, where we need to rely on a function approximator to represent the soft Q-values, and running the two steps until convergence would be computationally too expensive. The approximation gives rise to a new practical algorithm, called soft actor-critic.

Theorem 2 (Soft Policy Iteration). *Repeated application of soft policy evaluation and soft policy improvement from any $\pi \in \Pi$ converges to a policy π^* such that $Q^{\pi^*}(\mathbf{s}_t, \mathbf{a}_t) \geq Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ for all $\pi \in \Pi$ and $(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{S} \times \mathcal{A}$, assuming $|\mathcal{A}| < \infty$.*

Proof. See Appendix B.3. □

Soft Actor-Critic

As discussed above, large continuous domains require us to derive a practical approximation to soft policy iteration. To that end, we use function approximators for both the soft Q-function and the policy, and instead of running evaluation and improvement to convergence, alternate between optimizing both networks with stochastic gradient descent. We consider a parameterized Q-function $Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$ and a tractable policy $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$. The parameters of these networks are θ and ϕ. For example, the Q-function can be modeled as expressive neural networks, and the policy as a Gaussian with mean and covariance given by neural networks. We will next derive update rules for these parameter vectors.

The Q-function parameters can be trained to minimize the soft Bellman residual

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \left(r(\mathbf{s}_t,\mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar\theta}(\mathbf{s}_{t+1})\right]\right)\right)^2\right], \tag{4.4}$$

where the value function is implicitly parameterized through the Q-function parameters via Equation 4.2,[1] and it can be optimized with stochastic gradients

$$\hat\nabla_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t,\mathbf{s}_t)\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \left(r(\mathbf{s}_t,\mathbf{a}_t) + \gamma\left(Q_{\bar\theta}(\mathbf{s}_{t+1},\mathbf{a}_{t+1}) - \alpha\log\left(\pi_\phi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})\right)\right)\right)\right). \tag{4.5}$$

The update makes use of a target Q-function with parameters $\bar\theta$ that are obtained as an exponentially moving average of the Q-function weights, which has been shown to stabilize training (Mnih et al., 2015). Alternatively, we can update the target weights to match the target function weights periodically (see Section D.2). Finally, the policy parameters can be learned by directly minimizing the expected KL-divergence in Equation 4.3:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\mathrm{D}_{\mathrm{KL}}\left(\pi_\phi(\,\cdot\,|\mathbf{s}_t)\,\middle\|\,\frac{\exp\left(\frac{1}{\alpha}Q_\theta(\mathbf{s}_t,\,\cdot\,)\right)}{Z_\theta(\mathbf{s}_t)}\right)\right]. \tag{4.6}$$

There are two options for minimizing J_π. A typical solution for policy gradient methods is to use the likelihood ratio gradient estimator (Williams, 1992), which does not require backpropagating the gradient through the policy and the target density networks. However, in our case, the target density is the soft Q-function, which is represented by a neural network an can be differentiated, and it is thus convenient to apply the reparameterization trick instead, resulting in a lower variance estimator. To that end, we reparameterize the policy using a neural network transformation

$$\mathbf{a}_t = f_\phi(\epsilon_t;\mathbf{s}_t), \tag{4.7}$$

where ϵ_t is an input noise vector, sampled from some fixed distribution, such as a spherical Gaussian. We can now rewrite the objective in Equation 4.6 as

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\alpha\log\pi_\phi(f_\phi(\epsilon_t;\mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t;\mathbf{s}_t))\right], \tag{4.8}$$

[1] In (Haarnoja et al., 2018c) we introduced an additional function approximator for the value function, but later found it to be unnecessary.

Algorithm 2 Soft Actor-Critic

```
Input: $\theta_1$, $\theta_2$, $\phi$                                          ▷ Initial parameters
  $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$            ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$                                         ▷ Initialize an empty replay pool
  for each iteration do
    for each environment step do
      $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$                          ▷ Sample action from the policy
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$            ▷ Sample transition from the environment
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$    ▷ Store the transition in the replay pool
    end for
    for each gradient step do
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$     ▷ Update the soft Q-function parameters
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$                  ▷ Update policy weights
      $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$       ▷ Update target network weights
    end for
  end for
Output: $\theta_1$, $\theta_2$, $\phi$                                         ▷ Optimized parameters
```

where π_ϕ is defined implicitly in terms of f_ϕ, and we have noted that the partition function is independent of ϕ and can thus be omitted. We can approximate the gradient of Equation 4.8 with

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \alpha \log\left(\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right) + \left(\nabla_{\mathbf{a}_t} \alpha \log\left(\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right) - \nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t)\right) \nabla_\phi f_\phi(\epsilon_t; \mathbf{s}_t), \tag{4.9}$$

where $\mathbf{a}_t$ is evaluated at $f_\phi(\epsilon_t; \mathbf{s}_t)$. This unbiased gradient estimator extends the DDPG style policy gradients (Lillicrap et al., 2015) to any tractable stochastic policy.

Our algorithm also makes use of two Q-functions to mitigate positive bias in the policy improvement step that is known to degrade performance of value based methods (Hasselt, 2010; Fujimoto et al., 2018). In particular, we parameterize two Q-functions, with parameters θ_i, and train them independently to optimize $J_Q(\theta_i)$. We then use the minimum of the Q-functions for the stochastic gradient in Equation 4.5 and policy gradient in Equation 4.9, as proposed by Fujimoto et al. (2018). Although our algorithm can learn challenging tasks, including a 21-dimensional Humanoid, using just a single Q-function, we found two Q-functions significantly speed up training, especially on harder tasks. The complete algorithm is described in Algorithm 2. The method alternates between collecting experience from the environment with the current policy and updating the function approximators using the stochastic gradients from batches sampled from a replay buffer. Using off-policy data from a replay buffer is feasible because both value estimators and the policy can be trained entirely on off-policy data. The algorithm is agnostic to the parameterization of the policy, as long as it can be evaluated for any arbitrary state-action tuple.

4.3 Experiments

The goal of our experimental evaluation is to understand how the sample complexity and stability of our method compares with prior off-policy and on-policy deep reinforcement learning algorithms. We compare our method to prior techniques on a range of challenging continuous control tasks from the OpenAI gym benchmark suite (Brockman et al., 2016) and also on the rllab implementation of the Humanoid task (Duan et al., 2016). Although the easier tasks can be solved by a wide range of different algorithms, the more complex benchmarks, such as the 21-dimensional Humanoid (rllab), are exceptionally difficult to solve with off-policy algorithms (Duan et al., 2016). The stability of the algorithm also plays a large role in performance: easier tasks make it more practical to tune hyperparameters to achieve good results, while the already narrow basins of effective hyperparameters become prohibitively small for the more sensitive algorithms on the hardest benchmarks, leading to poor performance (Gu et al., 2017).

We compare our method to deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), proximal policy optimization (PPO) (Schulman et al., 2017b), and soft Q-learning (SQL), which also optimizes the maximum entropy objective. We additionally compare to twin delayed deep deterministic policy gradient algorithm (TD3) (Fujimoto et al., 2018), using the author-provided implementation. This is a recent extension to DDPG that first applied the double Q-learning trick to continuous control along with other improvements. We have included a comparison to trust region path consistency learning (Trust-PCL) (Nachum et al., 2018) and two other variants of SAC in Section D.2. We turned off the exploration noise for evaluation for DDPG and PPO. For maximum entropy algorithms, which do not explicitly inject exploration noise, we either evaluated with the exploration noise (SQL) or use the mean action (SAC). The source code of our SAC implementation[2] and videos[3] are available online.

Comparative Evaluation

Figure 4.1 shows the total average return of evaluation rollouts during training for SAC, DDPG, PPO, SQL, and TD3. We train five different instances of each algorithm with different random seeds, with each performing one evaluation rollout every 1000 environment steps (with the exception of SQL, for which we show the training performance). The solid curves correspond to the mean and the shaded region to the minimum and maximum returns over the five trials. The results show that, overall, SAC performs comparably to the baseline methods on the easier tasks and outperforms them on the harder tasks with a large margin, both in terms of learning speed and the final performance. For example, DDPG fails to make any progress on Ant-v1, Humanoid-v1, and Humanoid (rllab), a result that is corroborated by prior work (Gu et al., 2017; Duan et al., 2016). Soft actor-critic also learns considerably faster than PPO as a consequence of the large batch sizes PPO needs to learn stably on more

[2]github.com/haarnoja/sac/
[3]sites.google.com/view/soft-actor-critic/

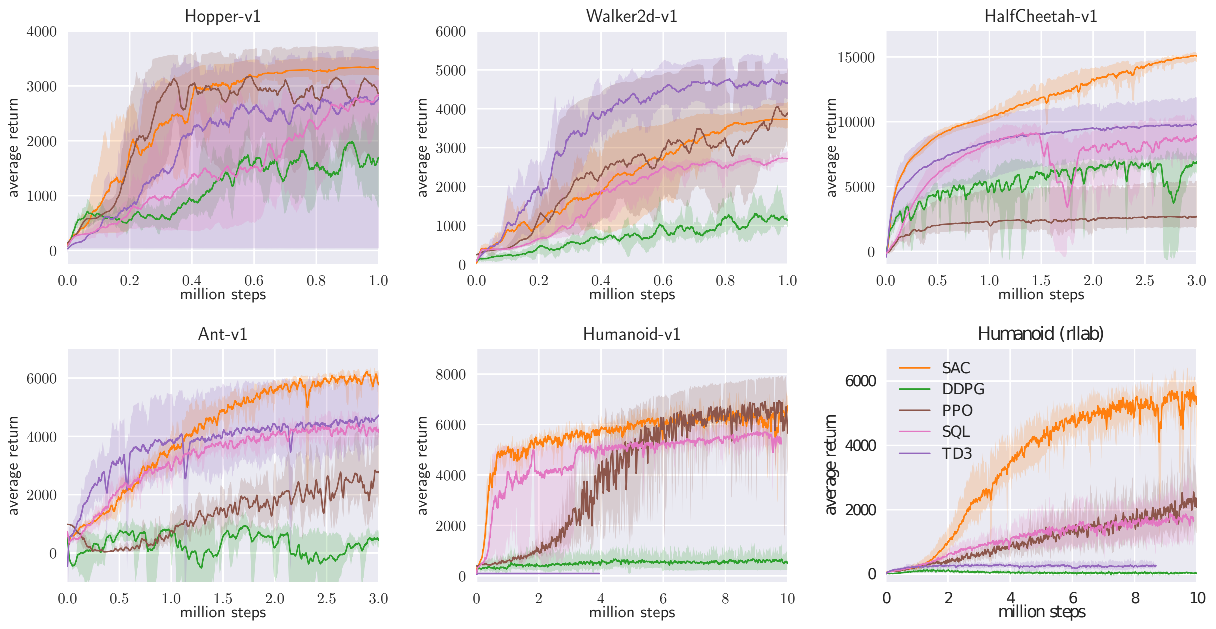


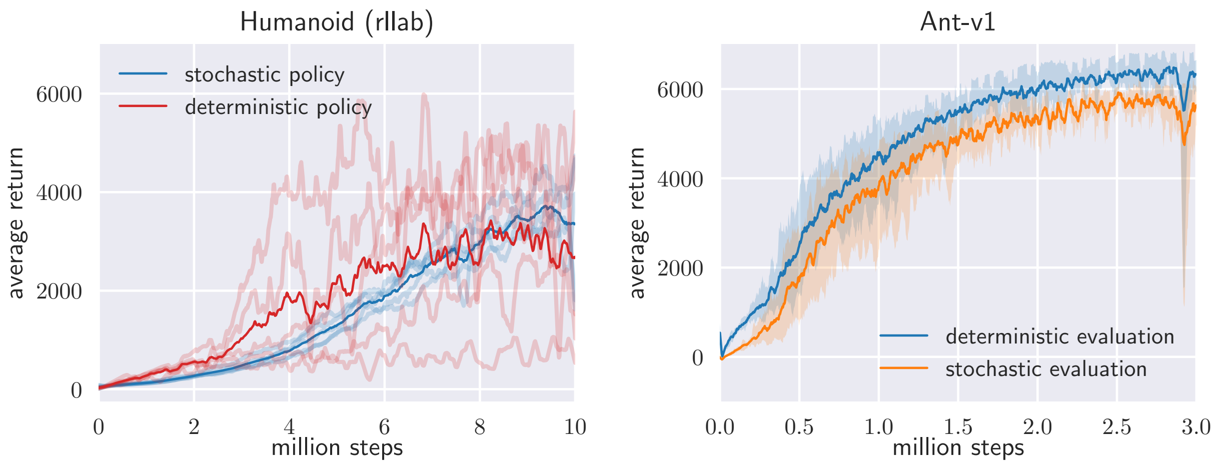
Figure 4.1: Training curves on continuous control benchmarks. Soft actor-critic (orange) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

high-dimensional and complex tasks. Soft Q-learning can also learn all tasks, but it is slower than SAC and has worse asymptotic performance. The quantitative results attained by SAC in our experiments also compare very favorably to results reported by other methods in prior work (Duan et al., 2016; Gu et al., 2017; Henderson et al., 2018), indicating that both the sample efficiency and final performance of SAC on these benchmark tasks exceeds the state of the art. All hyperparameters used in this experiment for SAC are listed in Appendix C.2.

Ablation Study

The results in the previous section suggest that algorithms based on the maximum entropy principle can outperform conventional RL methods on challenging tasks such as the humanoid tasks. In this section, we examine which particular components of SAC are important for good performance, and how sensitive SAC is to some of the most important hyperparameters, namely reward scaling and target value smoothing constant.

Stochastic vs. deterministic policy. Soft actor-critic learns stochastic policies via a maximum entropy objective. The entropy appears in both the policy and soft Q-function. In the policy, it prevents premature convergence of the policy variance. In the Q-function, it encourages exploration by increasing the value of regions of state space that lead to high-entropy behavior. To compare how the stochasticity of the policy and entropy maximiza-

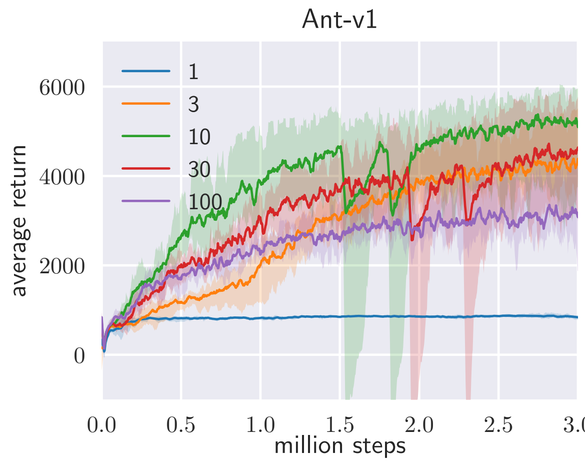


(a) (b)

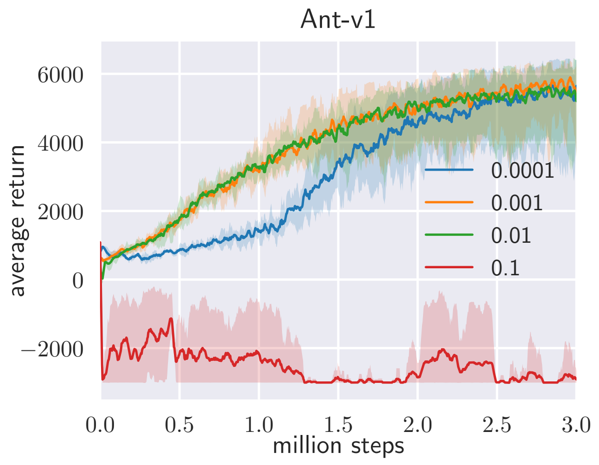
Figure 4.2: (a) Comparison of SAC (blue) and a deterministic variant of SAC (red) in terms of the stability of individual random seeds on the Humanoid (rllab) benchmark. The comparison indicates that stochasticity can stabilize training as the variability between the seeds becomes much higher with a deterministic policy. (b) Evaluating the policy using the mean action generally results in a higher return. Note that the policy is trained to maximize also the entropy, and the mean action does not, in general, correspond the optimal action for the maximum return objective.

tion affects the performance, we compare to a deterministic variant of SAC that does not maximize the entropy and that closely resembles DDPG, with the exception of having two Q-functions, using hard target updates, not having a separate target actor, and using fixed rather than learned exploration noise. Figure 4.2a compares five individual runs with both variants, initialized with different random seeds. Soft actor-critic performs much more consistently, while the deterministic variant exhibits very high variability across seeds, indicating substantially worse stability. As evident from the figure, learning a stochastic policy with entropy maximization can drastically stabilize training. This becomes especially important with harder tasks, where tuning hyperparameters is challenging. In this comparison, we updated the target value network weights with hard updates, by periodically overwriting the target network parameters to match the current value network (see Section D.2 for a comparison of average performance on all benchmark tasks).

Policy evaluation. Since SAC converges to stochastic policies, it is often beneficial to make the final policy deterministic at the end for best performance. For evaluation, we approximate the maximum a posteriori action by choosing the mean of the policy distribution. Figure 4.2b compares training returns to evaluation returns obtained with this strategy indicating that deterministic evaluation can yield better performance. It should be noted that



(a) Reward scale



(b) Target smoothing coefficient (τ)

Figure 4.3: Sensitivity of soft actor-critic to selected hyperparameters on Ant-v1 task. (a) Soft actor-critic is sensitive to reward scaling since it is related to the temperature of the optimal policy. The optimal reward scale varies between environments, and should be tuned for each task separately. (b) Target value smoothing coefficient τ is used to stabilize training. Fast moving target (large τ) can result in instabilities (red), whereas slow moving target (small τ) makes training slower (blue).

all of the training curves depict the sum of rewards, which is different from the objective optimized by SAC and other maximum entropy RL algorithms, including SQL and Trust-PCL, which maximize also the entropy of the policy.

Reward scale. Soft actor-critic is particularly sensitive to the scaling of the reward signal, because it directly affects the temperature of the energy-based optimal policy and thus controls its stochasticity. Larger reward magnitudes correspond to lower entropies. Figure 4.3a shows how learning performance changes when the reward scale is varied: For small reward magnitudes, the policy becomes nearly uniform, and consequently fails to exploit the reward signal, resulting in substantial degradation of performance. For large reward magnitudes, the model learns quickly at first, but the policy then becomes nearly deterministic, leading to poor local minima due to lack of adequate exploration. With the right reward scaling, the model balances exploration and exploitation, leading to faster learning and better asymptotic performance. In practice, we found reward scale to be the only hyperparameter that requires tuning, and its natural interpretation as the inverse of the temperature in the maximum entropy framework provides good intuition for how to adjust this parameter. In the next chapter, we propose an automated adjustment scheme for the temperature, which eliminates the sensitivity to the reward scale.

Target network update. It is common to use a separate target Q-value network that slowly tracks the actual Q-function to improve stability. We use an exponentially moving average, with a smoothing constant τ, to update the target Q-value network weights as common in the prior work (Lillicrap et al., 2015; Mnih et al., 2015). A value of one corresponds to a hard update where the weights are copied directly at every iteration and zero to not updating the target at all. In Figure 4.3b, we compare the performance of SAC when τ varies. Large τ can lead to instabilities while small τ can make training slower. However, we found the range of suitable values of τ to be relatively wide and we used the same value (0.005) across all of the tasks.

4.4 Conclusion

In this chapter, we presented soft actor-critic, an off-policy maximum entropy deep reinforcement learning algorithm that provides sample-efficient learning while retaining the benefits of entropy maximization and stability. Our theoretical results derive soft policy iteration, which we show to converge to the optimal policy. From this result, we can formulate a soft actor-critic algorithm, and we empirically show that it outperforms state-of-the-art model-free deep RL methods, including the off-policy DDPG algorithm and the on-policy PPO algorithm. In fact, the sample efficiency of this approach actually exceeds that of DDPG by a substantial margin. Our results suggest that stochastic, entropy maximizing reinforcement learning algorithms can provide a promising avenue for improved robustness and stability.

Chapter 5

Deep Reinforcement Learning with an Entropy Constraint

Even though maximum entropy RL algorithms, such as soft actor-critic, can improve sample efficiency of model-free RL, it introduces an additional temperature parameter that needs to be tuned for each task separately. This is particularly challenging in real-world application, since while hyperparameter tuning can be performed in parallel in simulated domains, it is usually impractical to tune hyperparameters directly on real-world platforms, especially for platforms that can easily damage themselves through extensive trial-and-error learning, such as legged robots. In this chapter, we develop an improvement to the soft actor-critic deep reinforcement learning algorithm that requires minimal hyperparameter tuning, while also requiring only a modest number of trials to learn multilayer neural network policies. This algorithm automatically trades off exploration against exploitation by dynamically and automatically tuning a temperature parameter that determines the stochasticity of the policy, and yet, can match the state-of-the-art performance on four standard benchmark environments. We then demonstrate that the algorithm can be used to learn quadrupedal locomotion gaits on a real-world Minitaur quadrupedal robot, learning directly in the real world in two hours.

5.1 Overview

One of the central challenges with the maximum entropy objective is that the trade-off between maximizing the return, or exploitation, versus the entropy, or exploration, is directly affected by the scale of the reward function. Unlike in conventional reinforcement learning, where the optimal policy is independent of scaling of the reward function, in maximum entropy reinforcement learning the scaling factor has to be compensated by the choice of a suitable temperature, and a sub-optimal temperature can drastically degrade the performance (Section 4.3). In this chapter, we propose a simple and automatic gradient-based temperature tuning method that adjusts the expected entropy over the visited states to match a target value. In contrast to standard reinforcement learning, our method only controls the expected entropy over states, while the per-state entropy can still vary—a desirable property that allows the policy to automatically reduce entropy for states where acting deterministically is preferred, while still acting stochastically in other states. Although this modification is simple, we find that in practice it virtually eliminates the need for per-task hyperparameter tuning, making it practical for us to apply this algorithm to learn quadrupedal locomotion gaits directly on a real-world robotic system.

The principal contributions of this chapter are an automatic temperature adjustment algorithm for maximum entropy reinforcement learning algorithms, such as soft actor-critic, and an empirical evaluation of this algorithm on both simulated benchmark tasks and on learning real-world quadrupedal locomotion on the Minitaur robot (Figure 5.6). In real-world experiments, our algorithm can enable the Minitaur to learn to walk using 160,000 environment steps in two hours without any hyperparameter adjustment. We also describe the details of our real-world reinforcement learning experimental setup. Furthermore, in evaluations on standard RL benchmark tasks, our method achieves state-of-the-art performance and, unlike prior state-of-the-art methods based on maximum entropy RL, can use exactly the same hyperparameters for all tasks. In the next section, we first derive a theoretical algorithm for finite horizon case, and then suggest a practical algorithm for discounted, infinite horizon case based on the theory.

5.2 Automating Entropy Adjustment for Maximum Entropy RL

In the previous chapter, we derived the soft actor-critic algorithm for learning maximum entropy policies of a given temperature. Unfortunately, choosing the optimal temperature is non-trivial, and the temperature needs to be tuned for each task. Instead of requiring the user to set the temperature manually, we can automate this process by formulating a different maximum entropy reinforcement learning objective, where the entropy is treated as a constraint. The magnitude of the reward differs not only across tasks, but it also depends on the policy, which improves over time during training. Since the optimal entropy depends on this magnitude, this makes the temperature adjustment particularly difficult: the entropy can vary unpredictably both across tasks and during training as the policy becomes better. Simply forcing the entropy to a fixed value is a poor solution, since the policy should be free to explore more in regions where the optimal action is uncertain, but remain more deterministic in states with a clear distinction between good and bad actions. Instead, we formulate a constrained optimization problem where the average entropy of the policy is constrained, while the entropy at different states can vary. Similar approach was taken in (Abdolmaleki et al., 2018), where the policy was constrained to remain close to the previous policy. We show that the dual to this constrained optimization leads to the soft actor-critic updates, along with an additional update for the dual variable, which plays the role of the temperature. Our formulation also makes it possible to learn the entropy with more expressive policies that can model multi-modal distributions, such as policies based on normalizing flows (Chapter 7) for which no closed form expression for the entropy exists. We derive the update for finite horizon case, and then derive an approximation for stationary policies by dropping the time dependencies from the policy, soft Q-function, and the temperature.

Constrained Entropy Objective

Our aim is to find a stochastic policy with maximal expected return that satisfies a minimum expected entropy constraint. Formally, we want to solve the constrained optimization problem

$$\max_{\pi_{0:T}} \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right] \text{ s.t. } \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[-\log(\pi_t(\mathbf{s}_t|\mathbf{s}_t))\right] \geq \bar{\mathcal{H}} \;\; \forall t \tag{5.1}$$

where $\mathcal{H}$ is a desired minimum expected entropy. Note that, for fully observed MDPs, the policy that optimizes the expected return is deterministic, so we expect this constraint to usually be tight and do not need to impose an upper bound on the entropy.

Since the policy at time t can only affect the future objective value, we can employ an (approximate) dynamic programming approach, solving for the policy backward through time. We rewrite the objective as an iterated maximization

$$\max_{\pi_0}\left(\mathbb{E}\left[r(\mathbf{s}_0, \mathbf{a}_0)\right] + \max_{\pi_1}\left(\mathbb{E}\left[\ldots\right] + \max_{\pi_T}\mathbb{E}\left[r(\mathbf{s}_T, \mathbf{a}_T)\right]\right)\right), \tag{5.2}$$

subject to the constraint on entropy. Starting from the last time step, we change the constrained maximization to the dual problem. Subject to $\mathbb{E}_{(\mathbf{s}_T,\mathbf{a}_T)\sim\rho_\pi}\left[-\log(\pi_T(\mathbf{s}_T|\mathbf{s}_T))\right] \geq \mathcal{H}$,

$$\max_{\pi_T}\mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r(\mathbf{s}_T, \mathbf{a}_T)\right] = \min_{\alpha_T\geq 0}\max_{\pi_T}\mathbb{E}\left[r(\mathbf{s}_T, \mathbf{a}_T) - \alpha_T\log\pi(\mathbf{a}_T|\mathbf{s}_T)\right] - \alpha_T\bar{\mathcal{H}}, \tag{5.3}$$

where α_T is the dual variable. We have also used strong duality, which holds since the objective is linear and the constraint (entropy) is convex function in π_T. This dual objective is closely related to the maximum entropy objective with respect to the policy, and the optimal policy is the maximum entropy policy corresponding to temperature α_T: $\pi^*_T(\mathbf{a}_T|\mathbf{s}_T;\alpha_T)$. We can solve for the optimal dual variable α^*_T as

$$\arg\min_{\alpha_T}\mathbb{E}_{\mathbf{s}_t,\mathbf{a}_t\sim\pi^*_t}\left[-\alpha_T\log\pi^*_T(\mathbf{a}_T|\mathbf{s}_T;\alpha_T) - \alpha_T\mathcal{H}\right]. \tag{5.4}$$

To simplify notation, we make use of the recursive definition of the soft Q-function

$$Q^*_t(\mathbf{s}_t, \mathbf{a}_t; \pi^*_{t+1:T}, \alpha^*_{t+1:T}) = \mathbb{E}\left[r(\mathbf{s}_t, \mathbf{a}_t)\right] + \mathbb{E}_{\rho_\pi}\left[Q^*_{t+1}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha^*_{t+1}\log\pi^*_{t+1}(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})\right], \tag{5.5}$$

with $Q^*_T(\mathbf{s}_T, \mathbf{a}_T) = \mathbb{E}\left[r(\mathbf{s}_T, \mathbf{a}_T)\right]$. Now, subject to the entropy constraints and again using the dual problem, we have

$$\begin{aligned}
&\max_{\pi_{T-1}} \left(\mathbb{E}\left[r(\mathbf{s}_{T-1}, \mathbf{a}_{T-1})\right] + \max_{\pi_T} \mathbb{E}\left[r(\mathbf{s}_T, \mathbf{a}_T)\right] \right) \qquad (5.6)\\
&\quad = \max_{\pi_{T-1}} \left(Q^*_{T-1}(\mathbf{s}_{T-1}, \mathbf{a}_{T-1}) - \alpha_T \bar{\mathcal{H}} \right)\\
&\quad = \min_{\alpha_{T-1} \geq 0} \max_{\pi_{T-1}} \left(\mathbb{E}\left[Q^*_{T-1}(\mathbf{s}_{T-1}, \mathbf{a}_{T-1})\right] - \mathbb{E}\left[\alpha_{T-1} \log \pi(\mathbf{a}_{T-1}|\mathbf{s}_{T-1})\right] - \alpha_{T-1}\bar{\mathcal{H}} \right) + \alpha^*_T \bar{\mathcal{H}}.
\end{aligned}$$

In this way, we can proceed backwards in time and recursively optimize Equation 5.1. Note that the optimal policy at time t is a function of the dual variable α_t. Similarly, we can solve the optimal dual variable α^*_t after solving for Q^*_t and π^*_t:

$$\alpha^*_t = \arg\min_{\alpha_t} \mathbb{E}_{\mathbf{a}_t \sim \pi^*_t} \left[-\alpha_t \log \pi^*_t(\mathbf{a}_t|\mathbf{s}_t; \alpha_t) - \alpha_t \bar{\mathcal{H}} \right]. \qquad (5.7)$$

The solution in Equation 5.7 along with the policy and soft Q-function updates described in Chapter 4 constitute the core of our algorithm, and in theory, exactly solving them recursively optimize the optimal entropy-constrained maximum expected return objective in Equation 5.1, but in practice, we will need to resort to function approximators and stochastic gradient descent.

Practical Algorithm

In principle, we can learn the soft Q-function and the policy using any maximum entropy reinforcement learning algorithm, and in practice we chose to use soft actor-critic as the base algorithm due to its good empirical performance and simplicity. In addition to the soft Q-function and the policy, we also learn α by minimizing the dual objective in Equation 5.7. This can be done by approximating dual gradient descent (Boyd & Vandenberghe, 2004). Dual gradient descent alternates between optimizing the Lagrangian with respect to the primal variables to convergence, and then taking a gradient step on the dual variables. While optimizing with respect to the primal variables fully is impractical, a truncated version that performs incomplete optimization (even for a single gradient step) can be shown to converge under convexity assumptions (Boyd & Vandenberghe, 2004). While such assumptions do not apply to the case of nonlinear function approximators such as neural networks, we found this approach to still work in practice. Thus, we compute gradients for α with the following objective:

$$J(\alpha) = \mathbb{E}_{\mathbf{a}_t \sim \pi_t} \left[-\alpha \log \pi_t(\mathbf{a}_t|\mathbf{s}_t) - \alpha \bar{\mathcal{H}} \right]. \qquad (5.8)$$

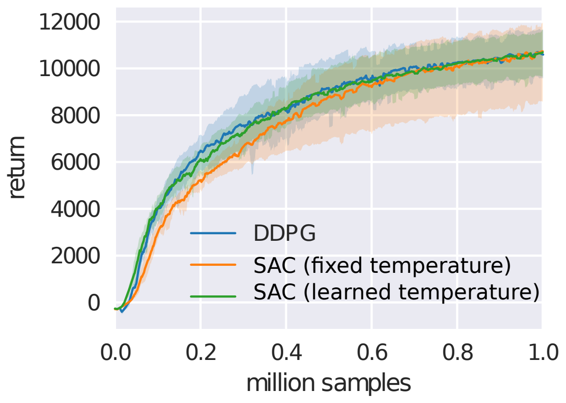
The proposed algorithm is listed in Algorithm 3, and it is otherwise identical to soft actor-critic, but it also includes an additional step for learning the temperature.

Algorithm 3 Soft Actor-Critic with Automatic Temperature Adjustment

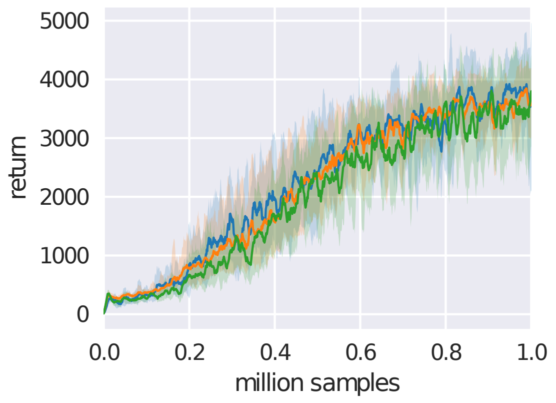
```
Input: $\theta_1$, $\theta_2$, $\phi$                                      ▷ Initial parameters
  $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$                       ▷ Initialize target network weights
  $\mathcal{D} \leftarrow \emptyset$                                        ▷ Initialize an empty replay pool
  for each iteration do
      for each environment step do
          $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$                        ▷ Sample action from the policy
          $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$             ▷ Sample transition from the environment
          $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$   ▷ Store the transition in the replay pool
      end for
      for each gradient step do
          $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$   ▷ Update the Q-function parameters
          $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$                ▷ Update policy weights
          $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$                  ▷ Adjust temperature
          $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$     ▷ Update target network weights
      end for
  end for
Output: $\theta_1$, $\theta_2$, $\phi$                                     ▷ Optimized parameters
```

5.3 Comparative Experiments in Simulation

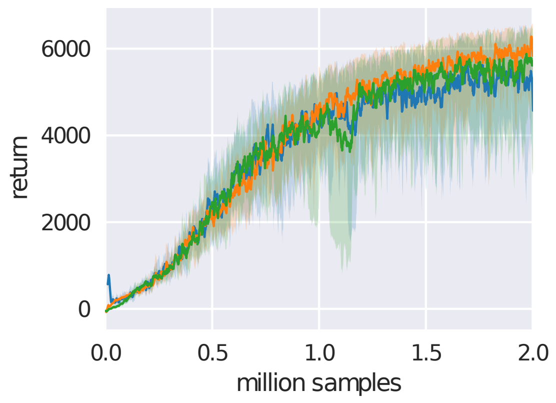
The aim of our simulated evaluation is to evaluate whether our algorithm can achieve state-of-the-art performance across a range of different domains without tuning of hyperparameters. To that end, we evaluate our algorithm on four continuous locomotion tasks from the Gym benchmark suite (Brockman et al., 2016), ranging from HalfCheetah-v1 with six action dimensions to Humanoid-v1 with 17-dimensional actions. We use SAC as the base algorithm and compare our automatic temperature adjustment scheme to a fixed temperature that is tuned for each environment separately using grid search. For our method, we simply set the target entropy to be -1 per action dimension (i.e. HalfCheetah-v1 has target entropy of -6, Humanoid-v1 uses -17). We also compare to deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015). Our method, as well as our version of DDPG, employ the modification suggested in (Fujimoto et al., 2018), namely learning two independent Q-functions and using the point-wise minimum as the target value for the critic and the policy updates, which has been shown to make learning substantially faster. All of the algorithms use the same network architecture: all of the function approximators (value function, policy, and Q-function for SAC) are parameterized with a two-layer neural network with 512 hidden units on each layer, and we use ADAM (Kingma & Ba, 2015) to learn the weights.



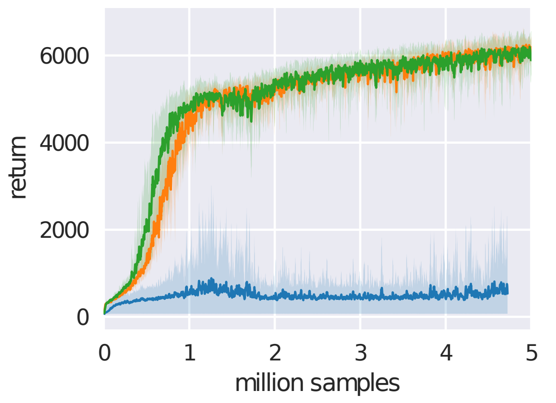
(a) HalfCheetah-v1



(b) Walker2d-v1



(c) Ant-v1

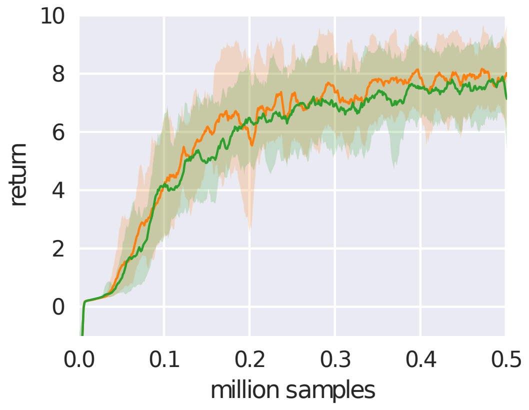


(d) Humanoid-v1

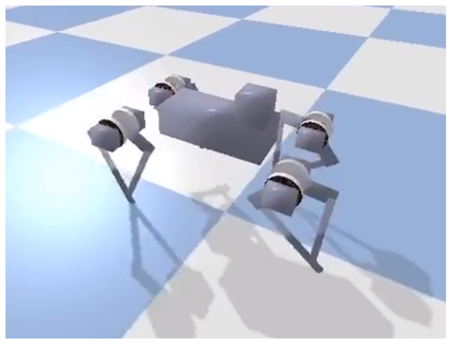
Figure 5.1: (a) – (d) Standard benchmark training results. Our method (green) is able to match the state-of-the-art baseline (orange) without environment specific tuning. DDPG (blue) performs well in lower-dimensional tasks, but fails on Humanoid-v1, which has 17 action dimensions.

Comparative Evaluation

Figure 5.1(a) – (d) shows a comparison of the algorithms. The solid line denotes the average performance over 10 random seeds, and the shaded region corresponds to the best and worst performing seeds. The results indicate that our method (green) achieves practically identical performance compared to SAC with a fixed temperature, which is tuned per environment for all environments (orange). We also compare to DDPG (blue), which does not require environment specific tuning. On the easiest three tasks, DDPG also achieves similar performance, but on a harder Humanoid-v1 task, DDPG fails to learn, which is consistent with



(a) Minitaur locomotion task



(b) Minitaur simulation

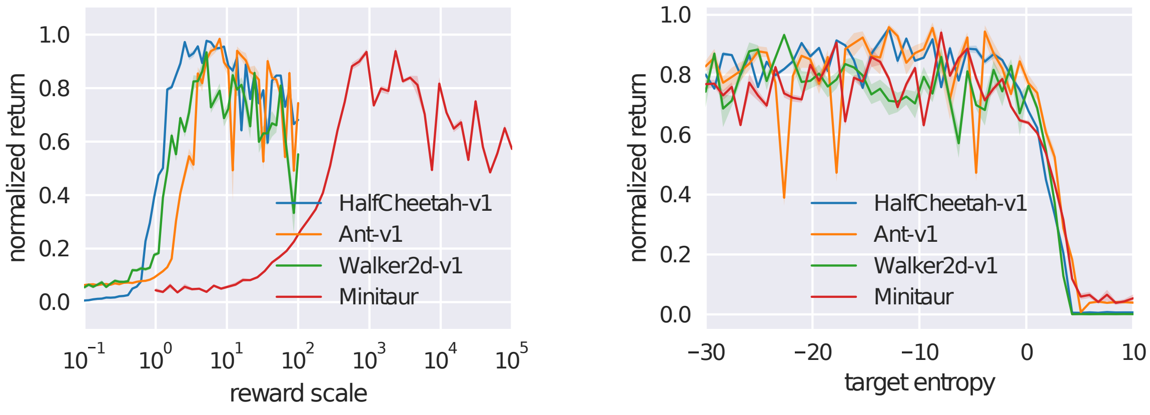
Figure 5.2: (a) Learning curves for the Minitaur environment. For our method (green), we used exactly the same hyperparameters as we used for the benchmarks whereas for the baseline (orange), we needed to tune the reward scale. (b) Illustration of the Minitaur environment.

the previously reported results (Henderson et al., 2018; Haarnoja et al., 2018c).

We also applied our method to a simulated Minitaur environment (Figure 5.2). Note that the purpose of testing in the simulated Minitaur environment is to demonstrate the insensitivity of hyperparameters of our algorithm. We do not use the policy learned in simulation in our real-world experiments. This Minitaur environment is more realistic than the benchmark environments, as it has been carefully system identified to match a real robotic platform: The learned controller in simulation can be transferred on the real Minitaur robot (Tan et al., 2018). Thus it is more representative of an actual use case of model-free reinforcement learning to robotics. Figure 5.2(a) compares the learning curve of our method (green), with exactly the same parameters used in the benchmarks, to SAC with fixed temperature (orange). Note that in order to obtain the baseline comparison, we had to collect approximately an order of magnitude more samples from the environment to tune the hyperparameter, which is not shown in the figure.

Sensitivity Analysis

In this section, we study the sensitivity of our method to reward scale and the value of the target entropy. Both maximum entropy RL algorithms (Chapter 4) and standard RL algorithms (Henderson et al., 2018) can be very sensitive to the scale of the reward function. In the case of maximum entropy RL, this scale directly affects the trade-off between reward maximization and entropy maximization (Haarnoja et al., 2018c). We evaluate sensitivity on the HalfCheetah-v1, Ant-v1, and Walker-v1 tasks, as well as the simulated Minitaur robot.



(a) Reward scale (b) Target entropy

Figure 5.3: Average normalized performance over the last 100k samples on a range of environments. (a) Performance of standard SAC as a function of reward scale, and (b) performance of the proposed method as a function of target entropy. Our method is substantially less sensitive to the choice of the hyperparameter controlling the stochasticity of the policy.

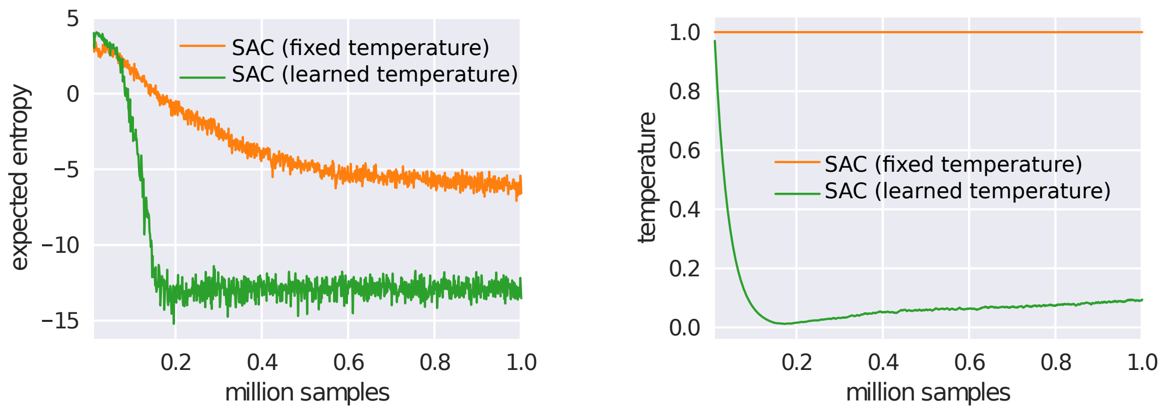
Figure 5.3a shows the normalized return for a range of reward scale values. All benchmark environments achieve good performance for about the same range of values, between 1 to 10. On the other hand, the simulated Minitaur requires roughly two orders of magnitude larger reward scale to work properly. This result indicates that, while standard benchmarks offer high variability in terms of task dimensionality, they are homogeneous in terms of other characteristics, and testing only on the benchmarks might not generalize well to seemingly similar tasks designed for different purposes. This suggests that the good performance of our method, with the same hyperparameters, on both the benchmark tasks and the Minitaur task accurately reflects its generality and robustness. Figure 5.3b compares the sensitivity of our method to the target entropy value on the same set of tasks. In this case, the range of good target entropy values is essentially the same for all environments, making hyperparameter tuning substantially less laborious. It is also worth noting that this range is very large, suggesting that our algorithm is relatively insensitive to the choice of this hyperparameter.

Next, we compared how the entropy and temperature evolve during training. Figure 5.4a compares the entropy (estimated as an expected negative log probability over a minibatch) on HalfCheetah-v1 for SAC with fixed temperature (orange) and our method (green). For our method, we used a target entropy of -13. The figure clearly indicates that our algorithm is able to match the target entropy in a relatively small number of steps. On the other hand, for SAC with a fixed temperature, the entropy slowly decreases as the Q-function increases. Figure 5.4b compares the temperature parameter of the two methods: Our method (green) actively adjust the temperature, and particularly in the beginning of training, when the Q-values are small and the entropy term thus dominates in the objective, temperature is quickly pulled down to match the target entropy.

Quadrupedal Locomotion in the Real World

In this section, we describe an application of our method to learn walking gaits directly in the real world. We use the Minitaur robot, a small-scale quadruped with eight direct-drive actuators (Kenneally et al., 2016). Each leg is controlled by two actuators that allow it to move in the sagittal plane. The Minitaur is equipped with motor encoders that measure the motor angles and an IMU that measures orientation and angular velocity of Minitaur's base. The action space are the swing angle and the extension of each leg, which are then mapped to desired motor positions and tracked with a PD controller (Tan et al., 2018). The observations include the motor angles as well as roll and pitch angles and angular velocities of the base. We exclude yaw since it is unreliable due to drift and irrelevant for the walking task. Note that latencies and contacts in the system make the dynamics non-Markovian, which can significantly degrade learning performance. We therefore construct the state out of the current and past five observations and actions. The reward function rewards large forward velocity, which is estimated using a motion capture system, and penalizes large angular accelerations, computed via finite differentiation from the last three actions. We also found it necessary to penalize for large pitch angles and for extending the front legs under the robot, which we found to be the most common failure cases that would require manual reset. We parameterize the policy and the value functions with feed-forward neural networks with two hidden-layers and 256 neurons per layer.

We developed a semi-automatic robot training pipeline (Figure 5.5) that consists of two parallel jobs: training and data collection. These jobs run asynchronously on two different



(a) Entropy (b) Temperature

Figure 5.4: Comparison our method and SAC with fixed temperature in terms of entropy and temperature on HalfCheetah-v1. The target entropy for our method is -13 in this case.

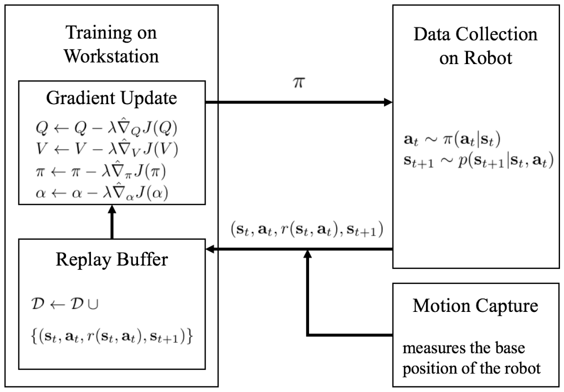

Training on Workstation
Gradient Update
$Q \leftarrow Q - \lambda\hat{\nabla}_Q J(Q)$
$V \leftarrow V - \lambda\hat{\nabla}_V J(V)$
$\pi \leftarrow \pi - \lambda\hat{\nabla}_\pi J(\pi)$
$\alpha \leftarrow \alpha - \lambda\hat{\nabla}_\alpha J(\alpha)$
Replay Buffer
$\mathcal{D} \leftarrow \mathcal{D} \cup$
$\{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
π
$(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})$
Data Collection on Robot
$\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
Motion Capture
measures the base position of the robot

Figure 5.5: The training pipeline runs the training and data collection asynchronously across multiple machines.

computers. The training process runs on a workstation, which updates the neural networks and periodically downloads the latest data from the robot and uploads the latest policy to the robot. On the robot, the on-board Nvidia Jetson TX2 runs the data collection job, which executes the policy, collects the trajectory and uploads these data to the workstation through Ethernet. Once the training is started, minimal human intervention is needed, except for the need to reset the robot state if it falls or drifts far from the initial state.

This learning task presents substantially challenges for real-world reinforcement learning. The robot is underactuated, and must therefore delicately balance contact forces on the legs to make forward progress. An untrained policy can lose balance and fall, and too many falls will eventually damage the robot, making sample-efficient learning essentially. Our method successfully learns to walk from 160k environment steps, or approximately 400 episodes with maximum length of 500 steps, which amount to approximately 2 hours of real-world training time.

However, in the real world, the utility of a locomotion policy hinges critically on its ability to generalize to different terrains and obstacles. Although we trained our policy only on flat terrain, as illustrated in Figure 5.6 (first row), we then tested it on varied terrains and obstacles (other rows). Because soft actor-critic learns robust policies, due to entropy maximization at training time, the policy can readily generalize to these perturbations without any additional learning. The robot is able to walk up and down a slope (first row), ram through an obstacle made of wooden blocks (second row), and step down stairs (third row) without difficulty, despite not being trained in these settings. To our knowledge, this experiment is the first example of a deep reinforcement learning algorithm learning underactuated quadrupedal locomotion directly in the real world without any simulation or pretraining. We have included videos of the the training process and evaluation on our project website[1].

[1]https://sites.google.com/view/sac-and-applications/

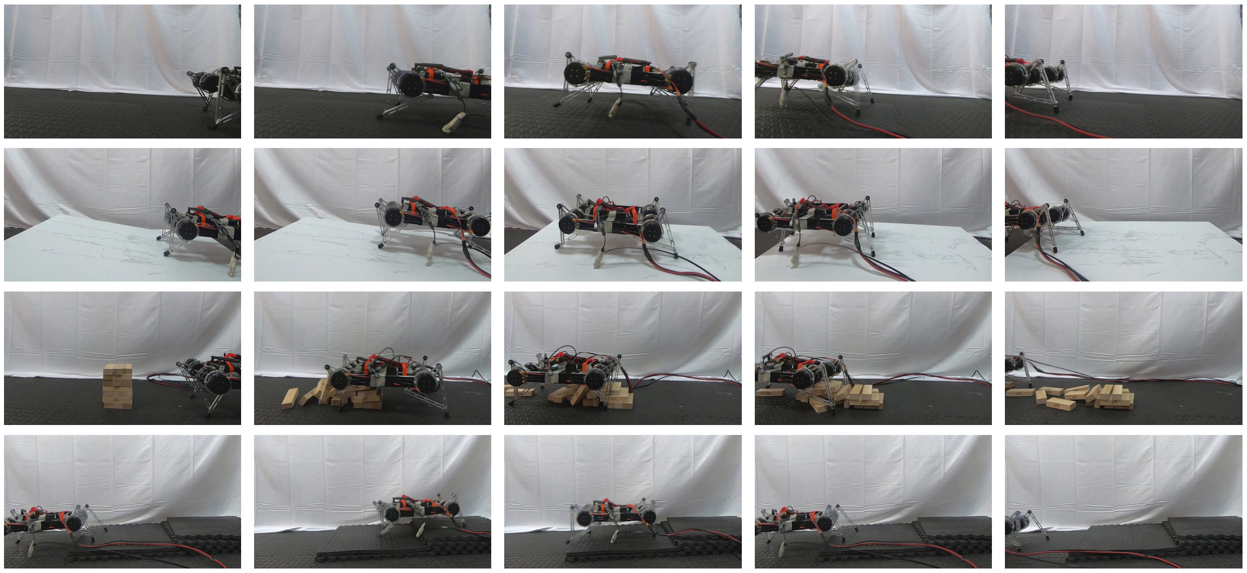


Figure 5.6: We trained the Minitaur robot to walk on flat terrain (first row) and tested how the learned gait generalizes to unseen situations (other rows)

5.4 Conclusion

We presented automatic entropy adjustment scheme for maximum entropy reinforcement learning, and implemented it with soft actor-critic. The resulting model-free reinforcement learning algorithm is sample-efficient and stable with respect to hyperparameter settings, which makes it well-suited for learning quadrupedal locomotion gaits end-to-end directly on a real-world Minitaur robot. Our method is based on a dual formulation of an entropy-constrained reinforcement learning objective, which builds on the framework of maximum entropy RL but introduces an explicit trade-off between exploration and exploitation that is invariant to reward magnitude and much easier to set in practice. The same entropy constraint value works well across all of the benchmark tasks we tested. Our method achieves state-of-the-art efficiency in simulated benchmarks, and can acquired a locomotion skill on real robot without any modification or tuning.

Part II

Extensions

Chapter 6

Compositionality of Maximum Entropy Policies

In this chapter, we show that policies learned with soft Q-learning can be composed to create new policies, and that the optimality of the resulting policy can be bounded in terms of the divergence between the policies at are combined. This compositionality provides a valuable tool for real-world manipulation, where constructing new policies by composing existing skills can provide a large gain in efficiency over training from scratch. Our experimental evaluation demonstrates that compositionality can be performed in both simulation and the real world.

6.1 Overview

The maximum entropy principle can yield an effective framework for practical, real-world deep reinforcement learning due to multiple reasons. In particular, as we saw in Chapter 3, maximum entropy policies provide an inherent, informed exploration strategy by expressing a stochastic policy via the Boltzmann distribution, with the energy corresponding to the reward-to-go or soft Q-function. This distribution assigns a non-zero probability to all actions, but actions with higher expected rewards are more likely to be sampled. As a consequence, the policy will automatically direct exploration into regions of higher expected return. This property, which can be thought of as a soft combination of exploration and exploitation, can be highly beneficial in real-world applications, since it provides considerably more structure than ϵ-greedy exploration and can thus improve sample complexity. In this section, we will focus on a closely related property, namely that independently trained maximum entropy policies can be composed together by adding their soft Q-functions, yielding a new policy for the combined reward function that is provably close to the corresponding optimal policy. Composability of controllers, which is not typically possible in standard reinforcement learning, is especially important for real-world applications, where reuse of past experience can greatly improve sample efficiency for tasks that can naturally be decomposed into simpler sub-problems. For instance, a policy for a pick-and-place task can be decomposed into (1) reaching specific x-coordinates, (2) reaching specific y-coordinates, and (3) avoiding certain obstacles. Such decomposable policies can therefore be learned in three stages, each yielding a sub-policy, which can later be combined offline without the need to interact with the environment.

The primary contribution of this chapter is a framework, based on soft Q-learning, for learning robotic manipulation skills with expressive neural network policies. We propose an extension of the SQL algorithm that enables composition of previously learned skills. We present a novel theoretical bound on the difference between the policy obtained through composition and the optimal policy for the composed reward function, which applies to SQL and other reinforcement learning methods based on maximum entropy objective. In our experiments, we leverage the compositionality of maximum entropy policies in both simulated and physical domains.

6.2 Compositionality of maximum entropy policies

Compositionality means that multiple policies can be combined to create a new policy that simultaneously solves all of the tasks given to the constituent policies. A related idea has been discussed by Todorov (2009), who considers combining the independent rewards via soft maximization. However, this type of composition corresponds to solving only one of the constituent tasks at a time—a kind of disjunction (e.g., move to the target *or* avoid the obstacle). In contrast, our approach to composition corresponds to a conjunction of the tasks, which is typically more useful (e.g., move to the target *and* avoid the obstacle). While simply adding soft Q-functions does not generally give the soft Q-function for the combined task, we show in this section that the regret from using the policy obtained by adding the constituent soft Q-functions together is upper bounded by the difference between the two policies that are being composed. Intuitively, if two composed policies agree on an action, or if they are indifferent towards each other's actions, then the composed policy will be closer to the optimal one.

Learning with Multiple Objectives

Compositionality makes learning far more efficient in multi-objective settings, which naturally emerge in robotic tasks. For example, when training a robot to move objects, one objective may be to move these objects quickly, and another objective may be to avoid collisions with a wall or a person. More generally, assume there are K tasks, defined by reward functions r_i for $i = 0, ..., K-1$, and that we are interested in solving any subset $\mathcal{C} \subseteq \{0, ..., K-1\}$ of those tasks simultaneously. A compound task can be expressed in terms of the sum of the individual rewards:

$$r_{\mathcal{C}}(\mathbf{s}, \mathbf{a}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} r_i(\mathbf{s}, \mathbf{a}). \tag{6.1}$$

The conventional approach is to solve the compound tasks by directly optimizing this compound reward $r_\mathcal{C}$ for every possible combination $\mathcal{C}$. Unfortunately, there are exponentially many combinations. Compositionality makes this learning process much faster by instead training an optimal policy π_i^* for each reward r_i and later combining them.

How should we combine the policies π_i^*? A simple approach is to approximate the optimal soft Q-function of the composed task $Q_\mathcal{C}^*$ with the mean of the individual soft Q-functions:

$$Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) \approx Q_\Sigma(\mathbf{s},\mathbf{a}) = \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}} Q_i^*(\mathbf{s},\mathbf{a}), \tag{6.2}$$

where Q_Σ represents an approximation to the true optimal soft Q-function of the composed task $Q_\mathcal{C}^*$. One can then extract a policy π_Σ from this approximate soft Q-function using any policy-extraction algorithm. In conventional reinforcement learning without entropy regularization, we cannot make any guarantees about how close Q_Σ is to $Q_\mathcal{C}^*$. However, we show in the next section that, if the constituent policies represent optimal maximum entropy policies, then we can bound the difference between the value of the approximate policy $Q_\mathcal{C}^{\pi_\Sigma}$ and the optimal value $Q_\mathcal{C}^*$ for the combined task.

Bounding the Sub-Optimality of Composed Policies

To understand what we can expect from the performance of the composed policy π_Σ that is induced by Q_Σ, we analyze how the value of π_Σ relates to the unknown optimal soft Q-function $Q_\mathcal{C}^*$ corresponding to the composed reward $r_\mathcal{C}$. For simplicity, we consider the special case where $\alpha = 1$ and we compose just two optimal policies, given by π_i^*, with soft Q-functions Q_i^* and reward functions r_i. Extending the proof to more than two policies and other values of α is straightforward. We start by introducing a lower bound for the optimal combined soft Q-function in terms of Q_i^* and π_i^*.

Lemma 5. *Let Q_1^* and Q_2^* be the soft Q-functions of the optimal policies corresponding to reward functions r_1 and r_2, and define $Q_\Sigma \triangleq \frac{1}{2}(Q_1^* + Q_2^*)$. Then the optimal soft Q-function of the combined reward $r_\mathcal{C} \triangleq \frac{1}{2}(r_1 + r_2)$ satisfies*

$$Q_\Sigma(\mathbf{s},\mathbf{a}) \geq Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) \geq Q_\Sigma(\mathbf{s},\mathbf{a}) - C^*(\mathbf{s},\mathbf{a}),\ \forall \mathbf{s}\in\mathcal{S}, \forall\mathbf{a}\in\mathcal{A}, \tag{6.3}$$

where C^* is the fixed point of

$$C(\mathbf{s},\mathbf{a}) \leftarrow \gamma\,\mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[\mathcal{D}_{\frac{1}{2}}\left(\pi_1^*(\,\cdot\,|\mathbf{s}') \parallel \pi_2^*(\,\cdot\,|\mathbf{s}')\right) + \max_{\mathbf{a}'\in\mathcal{A}} C(\mathbf{s}',\mathbf{a}')\right], \tag{6.4}$$

and $\mathcal{D}_{\frac{1}{2}}(\,\cdot \parallel \cdot\,)$ is the Rényi divergence of order $1/2$.

Proof. See Appendix A. ◻

This lower bound tells us that the simple additive composition of soft Q-functions never overestimates $Q^*_\mathcal{C}$ by more than the divergence of the constituent policies. Interestingly, the constant C^* can be obtained as the fixed point of the conventional Bellman equation, where the divergence of the constituent policies acts as the "reward function." Thus, C^* is the "value" of an adversarial policy that seeks to maximize the divergence. So far, we bounded $Q^*_\mathcal{C}$ in terms of the constituent soft Q-functions, which does not necessarily mean that the composed policy π_Σ will have a high value. To that end, we use soft policy evaluation (Section 4.2) to bound the value of the composed policy $Q^{\pi_\Sigma}_\mathcal{C}$:

Theorem 3. *With the definitions in Lemma 5, the value of π_Σ satisfies*

$$Q^{\pi_\Sigma}_\mathcal{C}(\mathbf{s}, \mathbf{a}) \geq Q^*_\mathcal{C}(\mathbf{s}, \mathbf{a}) - D^*(\mathbf{s}, \mathbf{a}), \tag{6.5}$$

where $D^*(\mathbf{s}, \mathbf{a})$ is the fixed point of

$$D(\mathbf{s}, \mathbf{a}) \leftarrow \gamma \, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathbb{E}_{\mathbf{a}' \sim \pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[C^*(\mathbf{s}', \mathbf{a}') + D(\mathbf{s}', \mathbf{a}') \right] \right]. \tag{6.6}$$

Proof. See Appendix B. □

Analogously to the discussion of C^* in Lemma 5, D^* can be viewed as the fixed point of a Bellman equation, where now $\gamma C^*(\mathbf{s}, \mathbf{a})$ has the role of the "reward function." Intuitively, this means that the bound becomes tight if the two policies agree in states that are visited by the composed policy. This result show that we can bound the regret from using the policy formed by composing optimal policies for the constituent rewards. While this bound is likely quite loose, it does indicate that the regret decreases as the divergence between the constituent policies decreases. This has a few interesting implications. First, deterministic policies always have infinite divergence, unless they are identical, which suggests that they are poorly suited for composition. Highly stochastic policies, such as those produced with maximum entropy algorithms like soft Q-learning, have much lower divergences. Intuitively, each policy has a wider distribution, and therefore the policies have more "overlap," thus reducing the regret from composition. Inversely, uniform policies always agree, and can thus be combined exactly. As a consequence, we expect maximum entropy RL methods to produce much more composable policies than standard RL methods.

6.3 Experiments

In this section, we show how compositionality of maximum entropy policies provides a practical tool for composing new compound skills out of previously trained components. We evaluate our method on a pushing task in simulation as well combined stacking while avoiding tasks on a real-world Sawyer robot. Videos of all experiments can be found on our website[1] and the code is available on GitHub[2].

[1] `sites.google.com/view/composing-real-world-policies/`

[2] `github.com/haarnoja/softqlearning/`

Experimental Setup

For the simulated tasks, we used the MuJoCo physics engine (Todorov et al., 2012), and for the real-world experiments we used the 7-DoF Sawyer robotic manipulator. The actions are the torque commands at each joint, and the observations are the joint angles and angular velocities, as well as the end-effector position. For the simulated pushing tasks, we also include all the relevant object coordinates, and for the experiments on Sawyer, we include the end-effector position and forces estimated from the actuator currents as observations. We parameterize the soft Q-function and the policies with a 2-layer neural network with 100 or 200 units in each layer and rectifier linear activations. We compare our method to deep deterministic policy gradients (DDPG) and normalized advangage functions (NAF).

Composing Policies for Pushing in Simulation

Does composing soft Q-functions from individual constituent policies allow us to quickly learn compound policies? To answer that question, we study the compositionality of policies in a simulated domain, where the task is to move a cylinder to a target location. This simulated evaluation allows us to provide a detailed comparison against prior methods, evaluating both their base performance and their performance under additive composition. We start by learning the soft Q-functions for the constituent tasks and then combining them by adding together the soft Q-functions to get Q_Σ. To extract a policy from Q_Σ, soft Q-learning requires training a policy to produce samples from $\exp(Q_\Sigma)$, and DDPG requires training a network that maximizes Q_Σ. In NAF, the optimal action for Q_Σ has a closed-form expression, since the constituent Q-functions are quadratic in the actions. We train the combined DDPG and SQL policies by reusing the data collected during the training of the constituent policies, thus imposing no additional sample cost.

The constituent tasks in our simulated evaluation require a planar arm to push a cylinder to specific positions along one of the two axes. A policy trained to push the disk to a specific x position can choose any arbitrary y position, and vice versa. A maximum entropy policy, in this case, would attempt to randomly cover all y positions, while a deterministic one would pick one arbitrarily. The composed policy is formed by combining a policy trained to push a cylinder to a specific x position and a policy trained to push the cylinder to a specific y location, thus solving a combined objective for moving the disk to a particular 2D location on the table. An illustration of this task is depicted in Figure 6.1, which shows a compound policy formed by combining a soft Q-function for pushing the disk the blue line ($x = -1$) and the orange line ($y = -1$). The final disk locations for 100 episodes of the final SQL policies are shown with dots of respective colors. The green dots illustrate the disk positions for the combined policy. Note that even though the orange policy never moves the disk to any point close to the intersection, the combined policy interpolates to the intended target correctly.

We trained four policies to push the cylinder to one of the following goals: left ($x = -1$), middle ($x = 0$), right ($x = 1$), and bottom ($y = -1$) using all three algorithm (SQL,

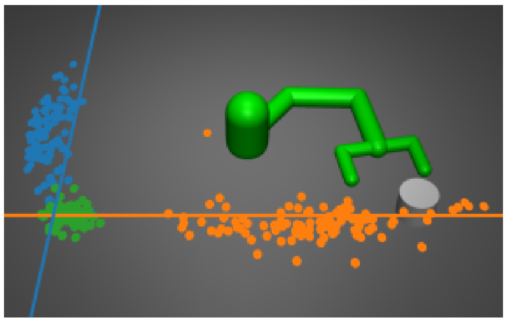


Figure 6.1: Two independent policies are trained to push the cylinder to the orange line and blue line, respectively. The colored dots show samples of the final location of the cylinder for the respective policies. When the policies are combined, the resulting policy pushes the cylinder to the lower intersection of the lines (green dots). No additional samples are used to train the combined policy. The combined policy learns to satisfy both original goals, rather than simply averaging the final cylinder location.

Table 6.1: Simulated Pushing Task: distance from cylinder to goal

Task	SQL	DDPG	NAF
push left	0.13 ± 0.06	0.20 ± 0.01	0.06 ± 0.01
push middle	0.07 ± 0.08	0.05 ± 0.03	0.02 ± 0.00
push right	0.10 ± 0.08	0.10 ± 0.07	0.09 ± 0.06
push bottom	0.08 ± 0.07	0.04 ± 0.02	0.17 ± 0.08
push bottom-left	0.11 ± 0.11	0.24 ± 0.01	0.17 ± 0.01
merge bottom-left	0.11 ± 0.05	0.19 ± 0.10	0.16 ± 0.00
push bottom-middle	0.12 ± 0.09	0.14 ± 0.03	0.34 ± 0.08
merge bottom-middle	0.09 ± 0.09	0.12 ± 0.02	0.02 ± 0.01
push bottom-right	0.18 ± 0.17	0.14 ± 0.06	0.15 ± 0.06
merge bottom-right	0.15 ± 0.14	0.09 ± 0.10	0.43 ± 0.21

DDPG, NAF). These goals gives three natural compositional objectives: bottom-left ($x = -1, y = -1$), bottom-middle ($x = 0, y = -1$), and bottom-right ($x = 1, y = -1$). Table 6.1 summarizes the error distance of each constituent policy (first four rows), policies trained to optimize the combined objective directly (e.g. , "push bottom-left"), and composed policies (e.g. , "merge bottom-left"). We see that SQL and DDPG perform well across all combined tasks, whereas NAF is able to combine some policies better (bottom-middle) but fails on others (bottom-right). Note that NAF policies are unimodal Gaussian, which are not well suited for compositions.

We also compared the different methods of learning a composed task in terms of training time. Example training curves for one of the targets are shown in Figure 6.2. Training a policy from a given Q_Σ (red, green) takes virtually no time for any of the algorithms when compared to training a Q-function and policy from scratch (blue). In fact, the combination of NAF policies has a closed form, and it is thus not included in the graph. For comparison, we

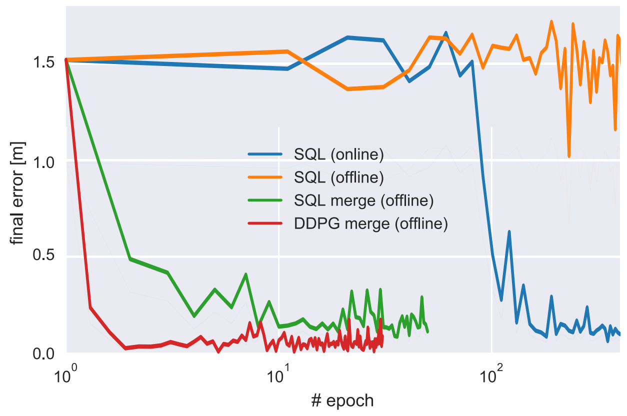


Figure 6.2: Comparison of the number of iterations needed to extract a policy from a Q-function (red and green) and training a policy using off-policy (orange) or on-policy data (blue). Note that the x-axis is on a log scale. Our proposed method (green and red) extracts a policy from an additively composed Q-function, Q_Σ, almost instantly in an offline fashion from off-policy data that were collected for training the constituent Q-functions. In contrast, training a policy from scratch with online SQL (blue) takes orders of magnitude more gradient steps and requires collecting new experience from the environment. Lastly, training a policy on the composed task using offline SQL with pre-collected data fails to converge in a reasonable amount of time (orange).

also trained a Q-function with SQL from scratch on the combined task in an offline fashion (orange), meaning that we only use samples collected by the policies trained on individual tasks. However, offline SQL fails to converge in reasonable time.

Our analysis indicates that, both with SQL and DDPG, compound policies can be obtained quickly and efficiently simply by adding together the soft Q-functions of the individual constituent policies. We can bound the suboptimality of such policies for SQL Section 6.2, but for DDPG our analysis does not apply since DDPG is the limiting case $\alpha = 0$. Nonetheless, on simple practical problems like the one in our experiments, even the DDPG Q-functions can be composed reasonably. This suggests that composition is a powerful tool that can allow us to efficiently build new skills out of old ones. Next, we will see that constituent policies trained with SQL can be reused to form compound skills also in the real world.

Composing Real-World Policies

To test the viability of compositionality in the real-world, we evaluated the compositionality of soft policies by combining a policy trained to stack Legos (Figure 6.3, first row) with a policy that avoids an obstacle (Figure 6.3, second row) with the Sawyer robot. The avoidance policy was rewarded for avoiding the obstacle and moving towards the target block without the shaping term that is required to successfully stack the two blocks. We then evaluated 1) the avoidance policy, 2) the stacking policy, and 3) the combined policy on the insertion task in the presence of the obstacle by executing each policy 10 times and counting the number of successful insertions. The avoidance policy was able to bring the Lego block close to the target block, but never successfully stacked the two blocks together. The stacking policy collided with the obstacle every time (Figure 6.3, third row), but was still able to solve the task 30 % of the time, and the combined policy (Figure 6.3, bottom row) successfully avoided the obstacle and stacked the blocks 100 % of the time. This experiment illustrates that policy compositionality is an effective tool that can be successfully employed also to real-world tasks.

6.4 Conclusion

We discussed how expressive maximum entropy policies trained with soft Q-learning can be composed to form new policies that are approximately optimal for the combined task, defined by the sum of the constituent reward functions. Soft Q-learning achieves substantially better performance than NAF on a simulated reaching task, where multiple policies are composed to reach to new locations. The ability to compose Q-functions obtained via soft Q-learning can make it particularly useful in real-world robotic scenarios, where retraining new policies for each new combination of reward factors is time-consuming and expensive.

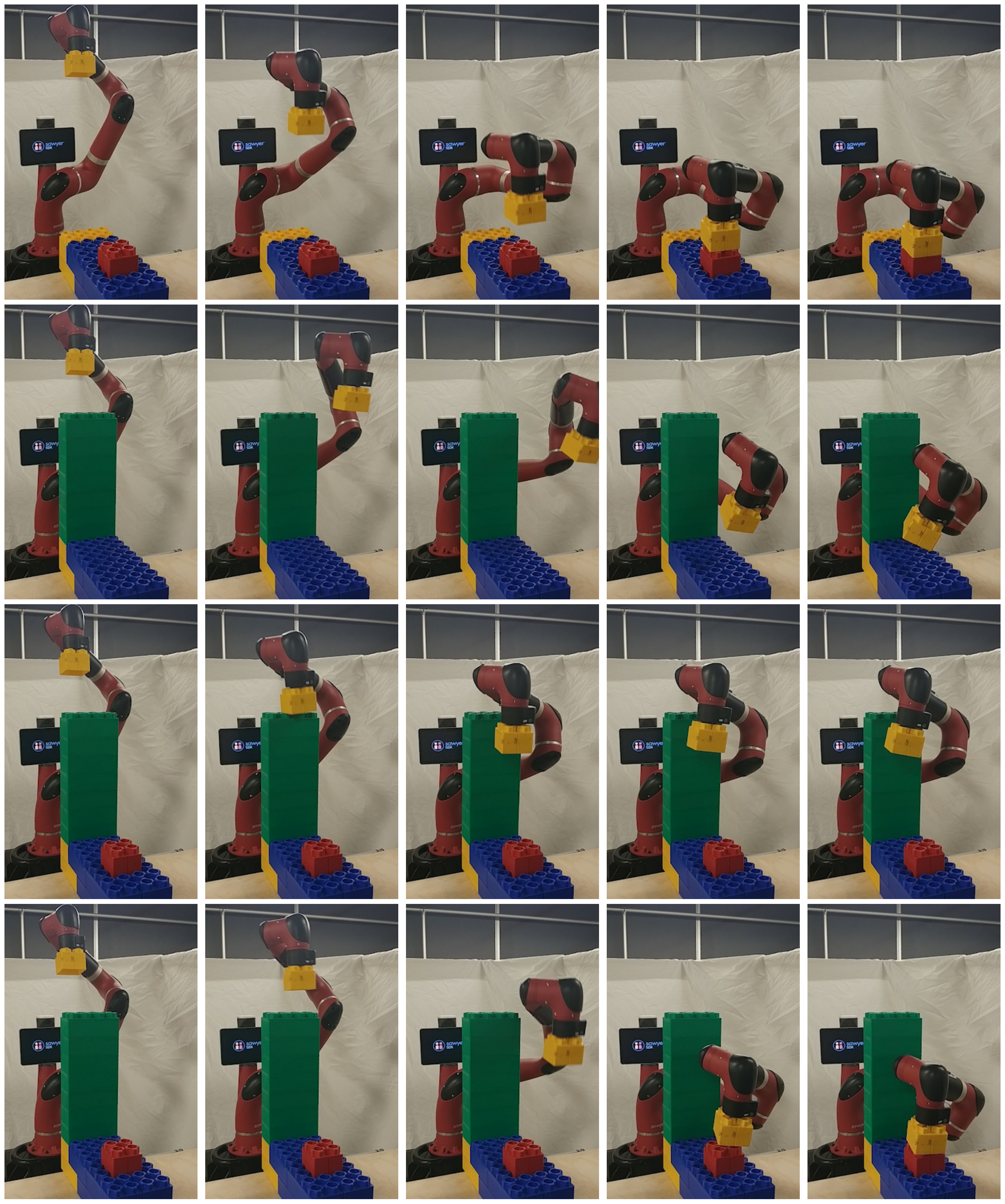


Figure 6.3: To illustrate compositionality on physical hardware, we trained the Sawyer manipulator to stack Lego blocks together (first row) and to avoid an obstacle (second row). The stacking policy fails if it is executed in the presence of the obstacle (third row). However, by combining the two policies, we get a new combined skills that solves both tasks simultaneously and is able to stack the blocks while avoiding the obstacle.

Chapter 7

Hierarchical Maximum Entropy Reinforcement Learning

In this chapter, we address the problem of learning hierarchical deep neural network policies for reinforcement learning. In contrast to methods that explicitly restrict or cripple lower layers of a hierarchy to force them to use higher-level modulating signals, each layer in our framework is trained to directly solve the task, but acquires a range of diverse strategies via a maximum entropy reinforcement learning objective. Each layer is also augmented with latent random variables, which are sampled from a prior distribution during the training of that layer. The maximum entropy objective causes these latent variables to be incorporated into the layer's policy, and the higher level layer can directly control the behavior of the lower layer through this latent space. Furthermore, by constraining the mapping from latent variables to actions to be invertible, higher layers retain full expressivity: neither the higher layers nor the lower layers are constrained in their behavior. Our experimental evaluation demonstrates that we can improve on the performance of single-layer policies on standard benchmark tasks simply by adding additional layers, and that our method can solve more complex sparse-reward tasks by learning higher-level policies on top of high-entropy skills optimized for simple low-level objectives.

7.1 Overview

Part of the promise of incorporating deep representations into RL is the potential for the emergence of hierarchies, which can enable reasoning and decision making at different levels of abstraction. A hierarchical RL algorithm could, in principle, efficiently discover solutions to complex problems and reuse representations between related tasks. While hierarchical structures have been observed to emerge in deep networks applied to perception tasks, such as computer vision (LeCun et al., 2015), it remains an open question how suitable hierarchical representations can be induced in a reinforcement learning setting. A central challenge with these methods is the automation of the hierarchy construction process: hand-specified hierarchies require considerable expertise and insight to design and limit the generality of the approach (Sutton et al., 1999; Kulkarni et al., 2016; Tessler et al., 2017), while automated methods must contend with severe challenges, such as the collapse of all primitives into just one useful skill (Bacon et al., 2017) or the need to hand-engineer primitive discovery objectives or intermediate goals (Heess et al., 2016).

When learning hierarchies automatically, we must answer a critical question: What objective can we use to ensure that lower layers in a hierarchy are useful to the higher layers? Prior work has proposed a number of heuristic approaches, such as hiding some parts of the observation from the lower layers (Heess et al., 2016), hand-designing state features and training the lower layer behaviors to maximize mutual information against these features (Florensa et al., 2017), or constructing diversity-seeking priors that cause lower-layer primitives to take on different roles (Daniel et al., 2012; Eysenbach et al., 2018). Oftentimes, these heuristics intentionally cripple the lower layers of the hierarchy, for example, by withholding information, so as to force a hierarchy to emerge, or else limit the higher levels of the hierarchy to selecting from among a discrete set of skills (Bacon et al., 2017). In both cases, the hierarchy is forced to emerge because neither higher nor lower layers can solve the problem alone. However, constraining the layers in this way can involve artificial and task-specific restrictions (Heess et al., 2016) or else diminish overall performance.

Instead of crippling or limiting the different levels of the hierarchy, we can imagine a hierarchical framework in which each layer directly attempts to solve the task and, if it is not fully successful, makes the job easier for the layer above it. In this paper, we explore a solution to the hierarchical reinforcement learning problem based on this principle. In our framework, each layer of the hierarchy corresponds to a policy with internal latent variables. These latent variables determine how the policy maps states into actions, and the latent variables of the lower-level policy act as the action space for the higher level. Crucially, each layer is unconstrained, both in its ability to sense and affect the environment: each layer receives the full state as the observation, and each layer is, by construction, fully invertible, so that higher layers in the hierarchy can undo any transformation of the action space imposed on the layers below.

In order to train policies with latent variables, we cast the problem of reinforcement learning into the framework of probabilistic graphical models. To that end, we build on maximum entropy reinforcement learning (Todorov, 2007; Ziebart et al., 2008), where the RL objective is modified to optimize for stochastic policies that maximize both reward and entropy. It can be shown that, in this framework, the RL problem becomes equivalent to an inference problem in a particular type of probabilistic graphical model (Toussaint, 2009). By augmenting this model with latent variables, we can derive a method that simultaneously produces a policy that attempts to solve the task, and a latent space that can be used by a higher-level controller to steer the policy's behavior.

The particular latent variable model representation that we use is based on normalizing flows (Dinh et al., 2016) that transform samples from a spherical Gaussian prior latent variable distribution into a posterior distribution, which in the case of our policies corresponds to a distribution over actions. When this transformation is described by a general-purpose neural network, the model can represent any distribution over the observed variable when the network is large enough. By conditioning the entire generation process on the state, we obtain a policy that can represent any conditional distribution over actions. When combined with maximum entropy reinforcement learning algorithms, this leads to a RL method that is expressive, powerful, and surprisingly stable. In fact, our experimental evaluation shows that this approach can attain state-of-the-art results on a number of continuous control benchmark tasks by itself, independently of its applicability to hierarchical RL.

The contributions of this chapter consist of a stable and scalable algorithm for training maximum entropy policies with latent variables as well as a framework for constructing hierarchies out of these latent variable policies. Hierarchies are constructed in layerwise fashion, by training one latent variable policy at a time, with each policy using the latent space of the policy below it as an action space, as illustrated in Figure 7.2. Each layer can be trained either on the true reward for the task, without any modification, or on a lower-level shaping reward. For example, for learning a complex navigation task, lower layers might receive a reward that promotes locomotion, regardless of direction, while higher layers aim to reach a particular location. When the shaping terms are not available, the same reward function can be used for each layer, and we still observe significant improvements from hierarchy. Our experimental evaluation illustrates that our method produces state-of-the-art results in terms of sample complexity on a variety of benchmark tasks, including a humanoid robot with 21 actuators, even when training a single layer of the hierarchy, and can further improve performance when additional layers are added. Furthermore, we illustrate that more challenging tasks with sparse reward signals can be solved effectively by providing shaping rewards to lower layers.

7.2 Control as Inference

In this section, we derive the maximum entropy objective by transforming the optimal control problem into an inference problem. Our proposed hierarchical framework will later build off of this probabilistic view of optimal control (Section 7.3).

Probabilistic Graphical Model for Control

Our derivation is based on the probabilistic graphical model in Figure 7.1a. This model is composed of factors for the dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and for an action prior $p(\mathbf{a}_t)$, which is typically taken to be a uniform distribution but, as we will discuss later, will be convenient to set to a Gaussian distribution in the hierarchical case. Because we are interested in inferring the optimal trajectory distribution under a given reward function, we attach to each state and action a binary random variable $\mathcal{O}_t$, or *optimality variable*, denoting whether the time step was "optimal." To solve the optimal control problem, we can now infer the posterior action distribution $\pi^*(\mathbf{a}_t|\mathbf{s}_t) = p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T} = \mathit{true})$, which simply states that an optimal action is such that the optimality variable is active for the current state and for all

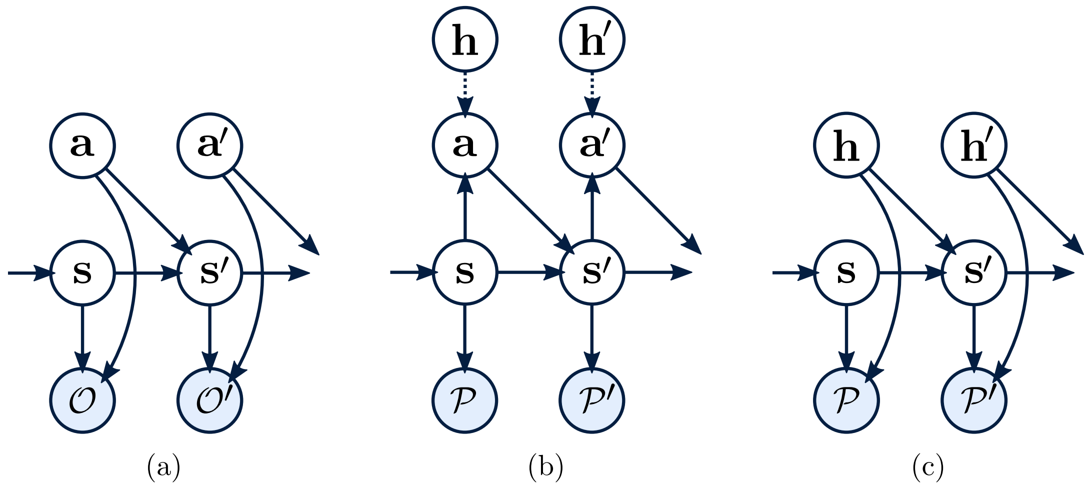


Figure 7.1: (a) The optimal control problem can be cast as an inference problem by considering a graphical model that consist of transition probabilities, action priors, and optimality variables. We can infer the optimal actions by conditioning on the optimality variables. (b) We can then train a latent variable policy to approximate the optimal actions, augment the graphical model with the policy's action distribution, and condition on a new set of optimality variables $\mathcal{P}_{0:T}$. Dashed line denotes a deterministic (invertible) edge. (c) By marginalizing out the actions $\mathbf{a}_t$, we are left with a new model that is structurally identical to the one in (a), where $\mathbf{h}_t$ has taken the role of the original actions.

of the future states. For the remainder of this paper, we will refrain, for conciseness, from explicitly writing $\mathcal{O}_t = true$, and instead write $\mathcal{O}_t$ to denote the state-action tuple for the corresponding time was optimal.

A convenient way to incorporate reward function into this framework is to assume that $r(\mathbf{s}_t, \mathbf{a}_t) < 0$, without loss of generality, and choose $p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) = \exp(\frac{1}{\alpha} r(\mathbf{s}_t, \mathbf{a}_t))$. We can now write the distribution over optimal trajectories as

$$p(\tau|\mathcal{O}_{0:T}) \propto p(\mathbf{s}_0) \prod_{t=0}^{T} p(\mathbf{a}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \exp\left(\frac{1}{\alpha} r(\mathbf{s}_t, \mathbf{a}_t)\right), \tag{7.1}$$

and use it to make queries, such as $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T})$. As we will discuss in the next section, using variational inference to determine $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T})$ reduces to the familiar maximum entropy reinforcement learning objective.

Reinforcement Learning via Variational Inference

The optimal action distribution inferred from Equation 7.1 cannot be directly used as a policy for two reasons. First, it would lead to an overly optimistic policy that assumes that the stochastic state transitions can also be modified to prefer optimal behavior, even though in practice, the agent has no control over the dynamics. Second, in continuous domains, the optimal policy is intractable and has to be approximated, for example by using a Gaussian distribution. We can correct both issues by using structured variational inference, where we approximate the posterior with a probabilistic model that constrains the dynamics and the policy. We constrain the dynamics in this distribution to be equal to the true dynamics, which we do not need to actually know in practice but can simply sample in model-free fashion, and constrain the policy to some parameterized distribution. This defines the variational distribution $q(\tau)$ as

$$q(\tau) = p(\mathbf{s}_0) \prod_{t=0}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), \tag{7.2}$$

where $p(\mathbf{s}_0)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ are the true initial state distribution and dynamics, and $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is the parameterized policy that we wish to learn. We can fit this distribution by maximizing the evidence lower bound (ELBO):

$$\log p(\mathcal{O}_{0:T}) \geq -\mathrm{D}_{\mathrm{KL}} \left(q(\tau) \parallel p(\mathcal{O}_{0:T}, \tau) \right). \tag{7.3}$$

Since the dynamics and initial state distributions in q and p match, it's straightforward to check that the right hand side can be optimized by maximizing

$$J(\pi) = \mathbb{E}_{\tau \sim \rho_\pi(\tau)} \left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathrm{D}_{\mathrm{KL}} \left(\pi(\,\cdot\,|\mathbf{s}_t) \parallel p(\,\cdot\,) \right) \right], \tag{7.4}$$

which, if we choose a uniform action prior, is exactly the maximum entropy objective up to a constant, and it can be optimized with any off-the-shelf entropy maximizing reinforcement learning algorithms (e.g., (Nachum et al., 2017; Schulman et al., 2017a; Haarnoja et al., 2017, 2018c)). Although the variational inference framework is not the only way to motivate the maximum entropy RL objective, it provides a convenient starting point for our method, which will augment the graphical model in Figure 7.1a with latent variables that can then be used to produce a hierarchy of policies, as discussed in the following section.

7.3 Learning Latent Space Policies

In this section, we discuss how the probabilistic view of RL can be employed in the construction of hierarchical policies, by augmenting the graphical model in Figure 7.1a with latent variables. We will also propose a particular way to parameterize the distribution over actions conditioned on these latent variables that is based on bijective transformations, which will provide us with a model amenable to stable and tractable training and the ability for higher levels in the hierarchy to fully invert the behavior of the lower layers, as we will discuss in this section. We will derive the method for two-layer hierarchies to simplify notation, but it can be easily generalized to arbitrarily deep hierarchies. In this work, we consider the bottom-up approach, where we first train a low-level policy, and then use it to provide a higher-level action space for a higher level policy that ideally can now solve an easier problem.

Latent Variable Policies for Hierarchical RL

We start constructing a hierarchy by defining a stochastic base policy as a latent variable model. In other words, we require the base policy to consist of two factors: a conditional action distribution $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_t)$, where $\mathbf{h}_t$ is a latent random variable, and a prior $p(\mathbf{h}_t)$. Actions can be sampled from this policy by first sampling $\mathbf{h}_t$ from the prior and then sampling an action conditioned on $\mathbf{h}_t$. Adding the latent variables $\mathbf{h}_t$ results in a new graphical model, which can now be conditioned on some new optimality variables $\mathcal{P}_t$ that can represent either the same task, or a different higher-level task, as shown in Figure 7.1b. In this new graphical model, the base policy is integrated into the transition structure of the MDP, which now exposes a new, higher-level set of actions $\mathbf{h}_t$. Insofar as the base policy succeeds in solving the task, partially or completely, learning a related task with $\mathbf{h}_t$ as the action should be substantially easier.

This "policy-augmented" graphical model has a semantically identical interpretation as the original graphical model in Figure 7.1a, where the combination of the transition model and the action conditional serves as a new (and likely easier) dynamical system. We can derive the dynamics model for the combined system by marginalizing out the actions: $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{h}_t) = \int_{\mathcal{A}} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{h}_t)d\mathbf{a}_t$, where $\mathbf{h}_t$ is a new, higher-level action and $p(\mathbf{h}_t)$ is its prior, as illustrated in Figure 7.1c. In other words, the base policy shapes the underlying dynamics of the system, ideally making it more easily controllable by a higher level policy. We can now learn a higher level policy for the latents by conditioning on new optimality variables. We can repeat this process multiple times by integrating each new policy level into the dynamics and learning a new higher-level policy on top of the previous policy's latent space, thus constructing an arbitrarily deep hierarchical policy representation. We will refer to these policy layers as sub-policies in the following sections. Similar layerwise training has been studied in the context of generative modeling with deep believe networks and was shown to improve optimization of deep architectures (Hinton & Salakhutdinov, 2006).

Practical Training of Latent Variable Policies

An essential choice in our method is the representation of sub-policies which, ideally, should be characterized by three properties. First, each sub-policy should be tractable. This is required, since we need to maximize the log-likelihood of good actions, which requires marginalizing out all latent variables. Second, the sub-policies should be expressive, so that they don't suppress the information flow from the higher levels to the lower levels. For example, a mixture of Gaussians as a sub-policy can only provide a limited number of behaviors for the higher levels, corresponding to the mixture elements, potentially crippling the ability of the higher layers to solve the task. Third, the conditional factor of each sub-policy should be deterministic, since the higher levels view it as a part of the environment, and additional implicit noise in the system can degrade their performance. Our approach to model the conditionals is based on bijective transformations of the latent variables into actions, which provides all of the aforementioned properties: the sub-policies are tractable, since marginalization reduces to single-point evaluation; they are expressive if represented via neural networks; and they are deterministic due to the bijective transformation. Specifically, we borrow from the recent advances in unsupervised learning based on real-valued non-volume preserving (real NVP) neural network transformations (Dinh et al., 2016). Our network differs from the original real NVP architecture in that we also condition the transformations on the current state or observation. Note that, even though the transformation from the latent to the action is bijective, it can depend on the observation in arbitrarily complex non-invertible ways, as we discuss in Section 7.4, providing our sub-policies with the requisite expressive power.

We can learn the parameters of these networks by utilizing the change of variables formula as discussed by Dinh et al. (2016), and summarized here for completeness. Let $\mathbf{a}_t = f_\phi(\mathbf{h}_t; \mathbf{s}_t)$ be a bijective transformation, parameterized by ϕ, and let $\mathbf{h}_t \in \mathbb{R}^{\dim(\mathcal{A})}$ be a random variable and $p(\mathbf{h}_t)$ its prior density. It is possible to express the density of $\mathbf{a}_t$ in terms of the prior and the Jacobian of the transformation by employing the change of variable formula as

$$\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) = p(\mathbf{h}_t) \left| \det\left(\frac{df_\phi(\mathbf{h}_t; \mathbf{s}_t)}{d\mathbf{h}_t} \right) \right|^{-1}. \tag{7.5}$$

Dinh et al. (2016) propose a particular kind of bijective transformation, which has a triangular Jacobian, simplifying the computation of the determinant to a product of its diagonal elements. The exact structure of these transformations is outside of the scope of this work, and we refer the reader to (Dinh et al., 2016) for a more detailed description. We can easily chain these transformations to form multi-level policies, and we can train them end-to-end as a single policy or layerwise as a hierarchical policy. As a consequence, the policy representation is agnostic to whether or not it was trained as a single expressive latent-variable policy or as a hierarchical policy consisting of several sub-policies, allowing us to choose the training method that best suits the problem at hand without the need to redesign the policy topology each time from scratch.

Reward Functions for Policy Hierarchies

The simplest way to construct a hierarchy out of latent variable policies is to train each layer in turn, then freeze its weights, and train a new layer that uses the lower layer's latent variables as an action space. In this procedure, each layer is trained on the same maximum entropy objective, and each layer simplifies the task for the layer above it. As we will show in Section 7.4, this procedure can provide substantial benefit on challenging and high-dimensional benchmark tasks.

However, for tasks that are more challenging, we can also naturally incorporate weak prior information into the training process by using underdefined heuristic reward functions for training the lower layers. In this approach, lower layers of the hierarchy are trained on reward functions that include shaping terms that elicit more desirable behaviors. For example, if we wish to learn a complex navigation task for a walking robot, the lower-layer objective might provide a reward for moving in any direction, while the higher layer is trained only on the primary objective of the task. Because our method uses bijective transformations, the higher layer can always undo any behavior in the lower layer if it is detrimental to task success, while lower layer objectives that correlate with task success can greatly simplify learning. Furthermore, since each layer still aims to maximize entropy, even a weak objective, such as a reward for motion in any direction (e.g., the norm of the velocity), will produce motion in many different directions that is controllable by the lower layer's latent variables. In our experiments, we will demonstrate how this approach can be used to solve a goal navigation task for a simulated, quadrupedal robot.

Algorithm Summary

We summarize the proposed algorithm in Algorithm 4. The algorithm begins by operating on the low-level actions in the environment according to the unknown system dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, where we use $\mathbf{h}_t^{(0)} = \mathbf{a}_t$ to denote the lowest-layer actions. The algorithm is also provided with an ordered set of K reward functions r_i, which can all represent the same task or different tasks, depending on the skill we want to learn: skills that naturally divide into primitive skills can benefit from specifying a different low-level objective, but for simpler tasks, such as locomotion, which do not naturally divide into primitives, we can train each sub-policy to optimize the same objective. In both cases, the last reward function r_{K-1} should correspond to the actual task we want to solve. The algorithm then chooses each r_i sequentially and learns a maximum entropy policy π_{ϕ_i}, represented by an invertible transformation $f_{\phi_i} : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$ and a prior $p(\mathbf{h}_t^{(i)})$, to optimize the corresponding variational inference objective in Equation 7.4. We use soft actor-critic (Chapter 4) to optimize the policy due to its robustness and good sample-efficiency, although other entropy maximizing RL algorithms can also be used. After each iteration, we embed the newly learned transformation f_i into the environment, which produces a new system dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{h}_t^{(i+1)})$ that can be used by the next layer. As before, we do not need an analytic form of these dynamics, only the ability to sample from them, which allows our algorithm to operate in the fully model-free setting.

7.4 Experiments

Our experiments were conducted on several continuous control benchmark tasks from the OpenAI Gym benchmark suite (Brockman et al., 2016). The aim of our experiments was to answer the following questions: (1) How well does our latent space policy learning method compare to prior reinforcement learning algorithms? (2) Can we attain improved performance from adding additional latent variable policy layers to a hierarchy, especially on challenging, high-dimensional tasks? (3) Can we solve more complex tasks by providing

Algorithm 4 Latent Space Policy Learning

```
Input: $\{r_0, r_1, ..., r_{K-1}\}$                                   ▷ Set of reward functions
Input: $\{\phi_0, \phi_1, ..., \phi_{K-1}\}$                          ▷ Initial policy parameters
  for $i = 0$ to $K-1$ do
    $\phi_i^* \leftarrow \arg\max_{\phi_i} \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{h}_t^{(i)}) \sim \rho_{\pi_{\phi_i}}} \left[ r_i(\mathbf{s}_t, \mathbf{h}_t^{(i)}) - \alpha \log \left( \pi_{\phi_i}(\mathbf{h}_t^{(i)} | \mathbf{s}_t) \right) \right]$   ▷ Soft actor-critic
    where
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$               ▷ Environment dynamics
      $\mathbf{a}_t = (f_{\phi_0} \circ f_{\phi_1} \circ \cdots \circ f_{\phi_{i-1}})(\mathbf{h}_t^{(i)})$   ▷ Lower level policies
      $\mathbf{h}_t^{(i)} = f_{\phi_i}(\mathbf{h}_t^{(i+1)})$                          ▷ Current policy
      $\mathbf{h}_t^{(i+1)} \sim p(\mathbf{h}_t^{(i+1)})$                           ▷ Latent prior (spherical Gaussian)
  end for
Output: $f^* = f_{\phi_0^*} \circ f_{\phi_1^*} \circ \cdots \circ f_{\phi_{K-1}^*}$        ▷ Optimal hierarchical policy
```

simple heuristic shaping to lower layers of the hierarchy, while the higher layers optimize the original task reward? Videos of our experiments are available online[1].

Policy Architecture

In our experiments, we used both single-level policies and hierarchical policies composed of two sub-policies as shown on the right in Figure 7.2. Each sub-policy has an identical structure, as depicted on the left in Figure 7.2. A sub-policy is constructed from two *coupling layers* that are connected using the alternating pattern described in (Dinh et al., 2016). Our implementation differs from (Dinh et al., 2016) only in that we condition the coupling layers on the observations, via an embedding performed by a two-layer fully-connected network. In practice, we concatenate the embedding vector with the latent input to each coupling layer. Note that our method requires only the path from the input latent to the output to be invertible, and the output can depend on the observation in arbitrarily complex ways. We have released our code for reproducibility.[2]

Benchmark Tasks with Single-Level Policies

We compare our method (SAC-LSP) to proximal policy optimization (PPO) (Schulman et al., 2017b) and deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015). We also include two algorithm that learn maximum entropy policies: soft Q-learning (SQL) (Haarnoja et al., 2017), which also learns a sampling network as part of the model, represented by an implicit density model, and soft actor-critic, using a Gaussian mixture model policy (SAC-

[1]https://sites.google.com/view/latent-space-deep-rl/

[2]https://github.com/haarnoja/sac/

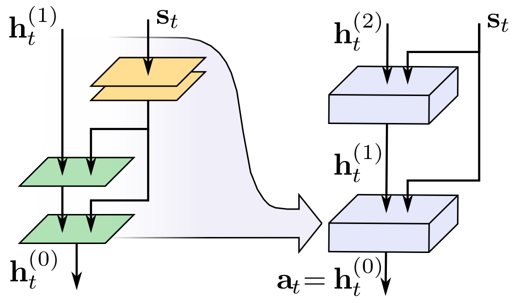


Figure 7.2: Our hierarchical policy consist of two levels (right diagram) that take in the observation and a latent vector from the previous level and outputs a latent vector to the next level. Diagram on the left shows the internal structure of each of the policy levels. The latent vector that is passed from the higher level is fed through two invertible coupling layers (green) (Dinh et al., 2016), which we condition on an observation embedding (yellow). Note that the path from the observation to the output does not need to be bijective, and therefore the observation embeddings can be represented with an arbitrary neural network, which in our case consists of two fully connected layers.

GMM)[3]. Note that the benchmark tasks compare the total expected return, but the entropy maximizing algorithms optimize a slightly different objective, so this comparison slightly favors PPO and DDPG, which optimize the benchmark objective directly. Another difference between the two classes of algorithms is that the maximum entropy policies are stochastic at test time, while DDPG is deterministic and PPO typically converges to nearly deterministic policies. For SAC-GMM, we execute an approximate maximum a posteriori action by choosing the mean of the mixture component that has the highest soft Q-value at test time, but for SQL and SAC-LSP, which both can represent an arbitrarily complex posterior distribution, we simply sample from the stochastic policy.

Our results on the benchmark tasks show that latent space policies based on normalizing flows generally performs on a par or better than all of the tested other methods, both in terms of efficiency and final return (Figure 7.3), especially on the more challenging and high-dimensional tasks, such as Humanoid (rllab). These results indicate that our policy representation can accelerate learning, particularly in challenging environments with high-dimensional actions and observations.

Multi-Level Policies

In this section, we evaluate the performance of our approach when we compose multiple latent variable policies into a hierarchy. In the first experiment, we train a single-level base policy on the most challenging standard benchmarks, Ant and Humanoid. We then freeze

[3]SAC-GMM is an early version of soft actor-critic that does not incorporate two Q-functions and uses REINFORCE gradients to estimate the gradient of the policy loss, and therefore performance is slightly worse than what we showed in Chapter 4 and Chapter 5.

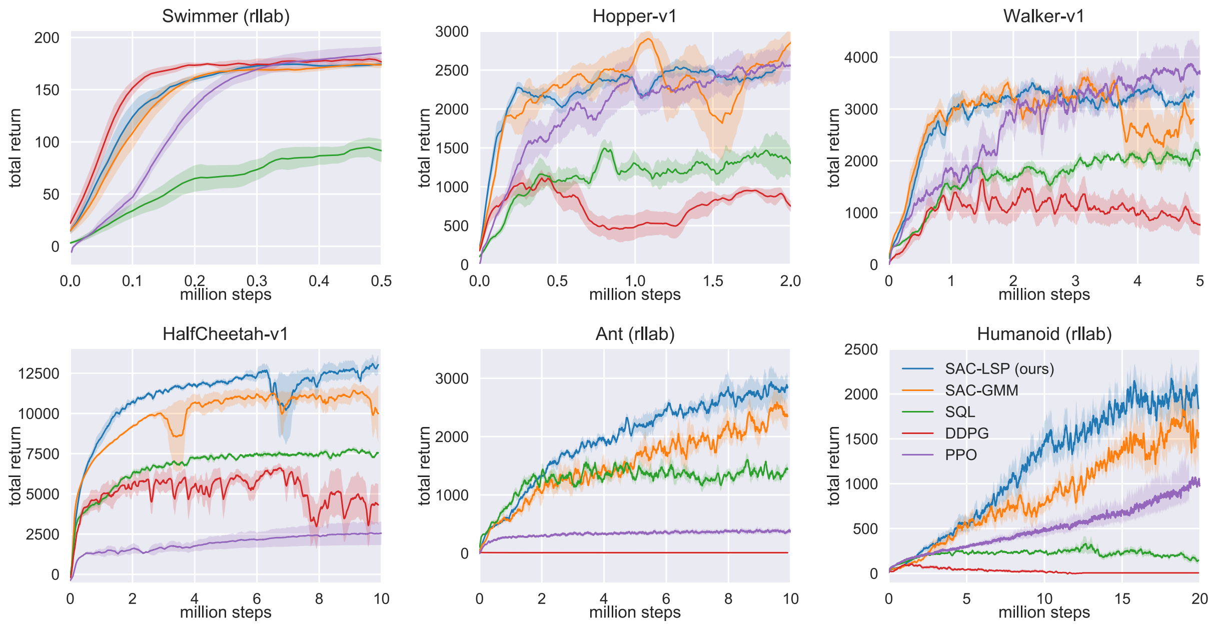
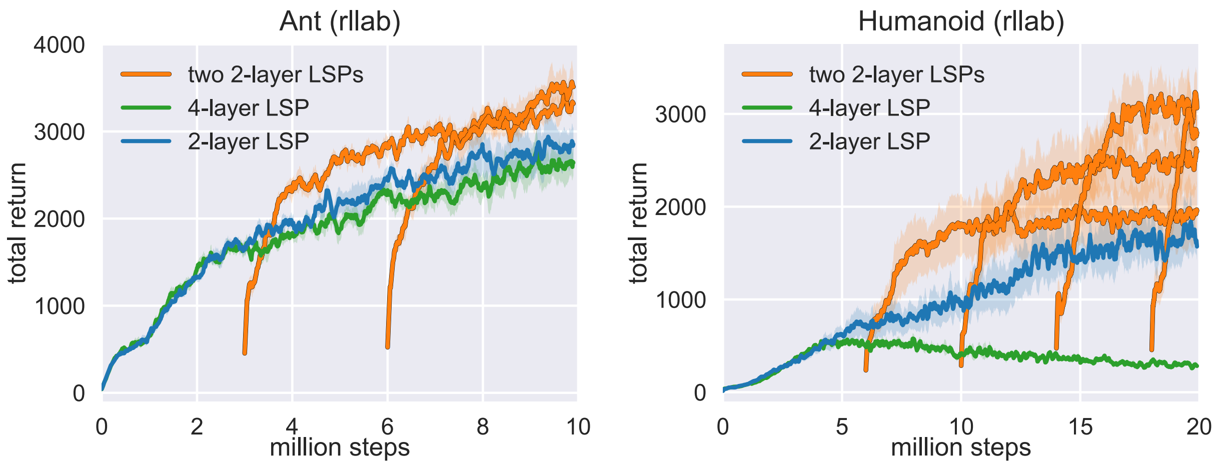


Figure 7.3: Training curves for continuous control benchmarks. Thick lines correspond to mean performance, and shaded regions show standard deviations of five random seeds. Our method (SAC-LSP) attains state-of-the-art performance across all tasks.

the weights of the base policy, and learn another policy level that uses the latent variables of the first policy as its action space. Intuitively, each layer in such a hierarchy attempts to solve the task to the best of its ability, providing an easier problem for the layer above. In Figure 7.4a and Figure 7.4b, we show the training curves for Ant and Humanoid, where the blue curve corresponds to the base policy and orange curves show the performance after we freeze the base policy weights and optimize a second-level policy. The different orange curves correspond to the addition of the second layer after a different numbers of training steps. In each case, the two-level policy can outperform a single level policy by a large margin. The performance boost is more prominent if we train the base policy longer. Note that the base policy corresponds to a single-level policy in Figure 7.3, and already learns more efficiently than prior methods. We also compare to a single, more expressive policy (green) that consists of four invertible layers, which has a similar number of parameters to a stacked two-level policy, but trained end-to-end as oppose to stagewise. This single four-layer policy performs comparably (Ant) or worse (Humanoid) than a single, two-layer policy (blue), indicating that the benefit of the two-level hierarchy is not just in the increased expressivity of the policy, but that the stagewise training procedure plays an important role in improving performance.

In our second experiment, we study how our method can be used to build hierarchical policies for complex tasks that require compound skills. The particular task we consider requires Ant to navigate through a simple maze (Figure 7.5a) with a sparse, binary reward

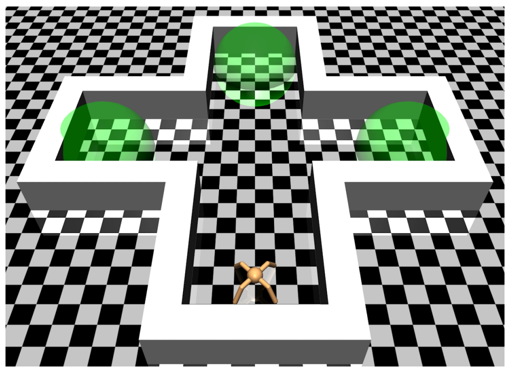


(a) Ant (rllab) (b) Humanoid (rllab)

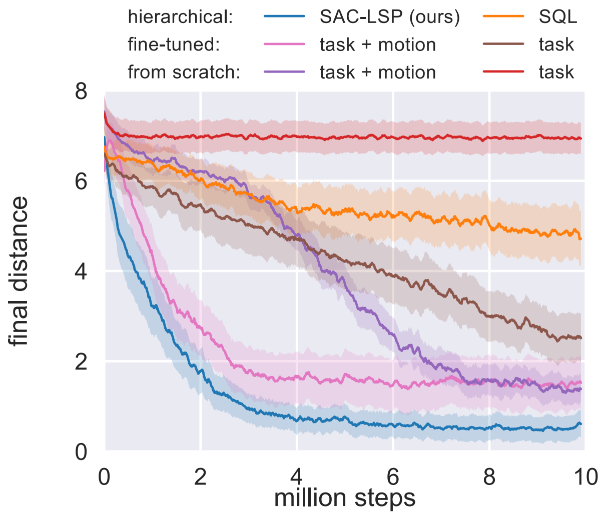
Figure 7.4: (a, b) We trained two-level policies for the most challenging benchmark tasks, Ant and Humanoid, by first training a single level policy (blue) and then freezing it and adding a second policy level. We repeated this procedure by starting training of the top-level policy at multiple points in time (orange). We also trained a single policy with four invertible layers end-to-end (green) for comparison. In each case, stagewise training of two levels yields the best performance.

for reaching the goals (shown in green). To solve this task, we can construct a hierarchy, where the lower layer is trained to acquire a general locomotion skill simply by providing a reward for maximizing velocity, regardless of direction. This pretraining phase can be conducted in a simpler, open environment where the agent can move freely. This provides a small amount of domain knowledge, but this type of domain knowledge is much easier to specify than true intermediate goals or modulation (Heess et al., 2016; Kulkarni et al., 2016), and the entropy maximization term in the objective automatically causes the low-level policy to learn a range of locomotion skills for various directions. The higher-level policy is then provided the standard task reward. Because the lower-level policy is invertible, the higher-level policy can still solve the task however it needs to, potentially even by fully undoing the behavior of the lower-level policy. However, the results suggest that the lower-level policy does indeed substantially simplify the problem, resulting in both rapid learning and good final performance.

In Figure 7.5b, we compare our approach (blue) to four single-level baselines that either learn a policy from scratch or fine-tune a pretrained policy. The pretraining phase (4 million steps) is not included in the learning curves, since the same base policies were reused across multiple tasks, corresponding to the three difference goal locations shown in Figure 7.5a. With task reward only, training a policy from scratch (red) failed to solve the task due to lack of structured exploration, whereas fine-tuning a pretrained policy (brown) that already knows how to move around and could occasionally find the way to the goal was able to



(a) Ant maze



(b) Ant maze results

Figure 7.5: (a) We trained Ant to navigate through a simple maze to three different goal location shown in green. (b) We first trained a low-level policy with a motion reward in a pretraining environment not including the walls, then fixed the policy and trained another policy level with a target reward (blue). We compare our method to learning a single policy from scratch or fine-tuning the pretrained policy using only the task reward or a combination of task and motion rewards. We also applied our method to soft Q-learning.

slowly learn this task. We also tried improving exploration by augmenting the objective with a motion reward, provided as a shaping term for the entire policy. In this case, a policy trained from scratch (purple) learned slowly, as it first needed to acquire a locomotion skill, but was able to eventually solve the task, while fine-tuning a pretrained policy resulted in much faster learning (pink). However, in both cases, adding motion as a shaping term prevents the policy from converging on an optimal solution, since the shaping alters the task. This manifests as residual error at convergence. On the other hand, our method (blue) can make use of the pretrained locomotion skills while optimizing for the task reward directly, resulting in faster learning and better final performance. Note that our method converges to a solution where the final distance to the goal is more than four times smaller than for the next best method, which finetunes with a shaped reward. We also applied our method to soft Q-learning (orange), which also trains latent space policies, but we found it to learn substantially slower than SAC-LSP.

7.5 Conclusion

We presented a method for training policies with latent variables. Our reinforcement learning algorithm not only compares favorably to state-of-the-art algorithms on standard benchmark tasks, but also provides an appealing avenue for constructing hierarchical policies: higher levels in the hierarchy can directly make use of the latent space of the lower levels as their action space, which allows us to train the entire hierarchy in a layerwise fashion. This approach to hierarchical reinforcement learning has a number of conceptual and practical benefits. First, each layer in the hierarchy can be trained with exactly the same algorithm. Second, by using an invertible mapping from latent variables to actions, each layer becomes invertible, which means that the higher layer can always perfectly invert any behavior of the lower layer. This makes it possible to train lower layers on heuristic shaping rewards, while higher layers can still optimize task-specific rewards with good asymptotic performance. Our method has a natural interpretation as an iterative procedure for constructing graphical models that gradually simplify the task dynamics.

Chapter 8

Discussion and Future Work

We presented deep RL methods based on the maximum entropy framework. We showed that policies learned under this framework can be composed to form new approximately optimal policies for the combination of the tasks, and we discussed how we can use it to build hierarchical policies. We also demonstrated that algorithms based on this framework, namely soft Q-learning and soft actor-critic, are sample efficient, provide consistent training performance, and are robust to variations hyperparameters—making them a strong candidate solution for real-world robotic tasks. However, several challenging to make deep RL a practical solution in the real world remain. Sample complexity is typically considered as the biggest, but given the recent advances of deep RL algorithms, we are getting to the point where we can solve simple tasks in the real world that do not require long term reasoning. Even though more can definitely be done on that front, there are a few other important aspects that have drawn less attention: reward specification and safe exploration.

Regarding the reward function, most advances in deep RL has focused on simulated systems or rule-based games, where specifying a reward signal is relatively straightforward. However, most real-world robotic applications require additional instrumentation, such as a motion capture system, adding substantial overhead to deployment of the algorithms and limiting their use to laboratories with access to the required instruments. On other hand, most real-world tasks are subjectively defined, and thus, a reasonable alternative to extensive instrumentation would be to design and implement improved ways to infer tasks directly from operators. For example, learning from human instructions (Tung et al., 2018), preferences (Christiano et al., 2017), or demonstrations (Abbeel & Ng, 2004) are all interesting and promising research directions. Regarding safety, maximum entropy RL can provide an elegant solution for safer exploration: Instead of maximizing entropy, that is, learning a posterior policy with a uniform prior, we can use a more informative prior distribution (Fox et al., 2016). An informative prior can be obtained via pretraining in simulation or from domain knowledge. A good prior equips the learning algorithm with an inductive bias for safer and better exploration, and the optimal policy can be eventually recovered by annealing the prior towards uniform.

Hierarchical reinforcement learning holds the promise of extending deep RL to tasks requiring longer term abstract reasoning. Policy hierarchies based on the maximum entropy framework and invertible policies yield a competitive performance and has an compelling probabilistic interpretation. The results we presented in Section 7.4 are limited to a single time scale[1], but in principle the framework could be extended to multiple time scales by imposing a slowness prior to the high-level policies. Another interesting research direction would be to study sim-to-real transfer with hierarchies: we could learn a base policy in simulation and a high-level policy in the real world. The high-level policy can be made less expressive to reduce the amount of required real-world samples, and since these policies are invertible, it can, in principle, learn to invert any undesirable behaviors learned by the base policy due to imperfect simulation model.

[1]To be precise, we used twice as long time steps for the high-level policy as for the base policy, but ideally the ratio should not be fixed a priori.

Bibliography

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5048–5058, 2017.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1726–1734, 2017.

Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 834–846, 1983.

Bateux, Q., Marchand, E., Leitner, J., Chaumette, F., and Corke, P. Visual servoing from deep neural networks. In *Robotics: Science and Systems (RSS), Workshop New Frontiers for Deep Learning in Robotics*, pp. 1–6, 2017.

Berseth, G., Xie, C., Cernek, P., and Van de Panne, M. Progressive reinforcement learning with distillation for multi-skilled motion control. *International Conference on Learning Representations (ICLR)*, 2018.

Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S., Maei, H. R., and Szepesvári, C. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1204–1212, 2009.

Boyd, S. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. P. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76 (1-2):5–23, 2016.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4299–4307, 2017.

Daniel, C., Neumann, G., and Peters, J. Hierarchical relative entropy policy search. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 273–281, 2012.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. In *International Conference on Neural Information Processing (NeurIPS), Deep Learning Symposium*, 2016.

Duan, Y., Chen, X. Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning (ICML)*, 2016.

Elfwing, S., Otsuka, M., Uchibe, E., and Doya, K. Free-energy based reinforcement learning for vision-based navigation with high-dimensional sensory inputs. In *International Conference on Neural Information Processing (NeurIPS)*, pp. 215–222, 2010.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Florensa, C., Duan, Y., and P., A. Stochastic neural networks for hierarchical reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Fox, R., Pakman, A., and Tishby, N. Taming the noise in reinforcement learning via soft updates. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2016.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.

Ghadirzadeh, A., Maki, A., Kragic, D., and Björkman, M. Deep predictive policy training using reinforcement learning. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 2351–2358. IEEE, 2017.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016. `http://www.deeplearningbook.org`.

Gruslys, A., Azar, M. G., Bellemare, M. G., and Munos, R. The reactor: A sample-efficient actor-critic architecture. *arXiv preprint arXiv:1704.04651*, 2017.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning (ICML)*, pp. 2829–2838, 2016.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *International Conference on Learning Representations (ICLR)*, 2017.

Guenter, F., Hersch, M., Calinon, S., and Billard, A. Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics*, 21(13):1521–1544, 2007.

Ha, S., Kim, J., and Yamane, K. Automated deep reinforcement learning environment for hardware of a modular legged robot. In *International Conference on Ubiquitous Robots (UR)*, pp. 348–354. IEEE, 2018.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, pp. 1352–1361, 2017.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018a.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018b.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018c.

Haarnoja, T., Zhou, A., Tan, J., Tucker, G., and Levine, S. End-to-end learning of quadrupedal locomotion with entropy constrained soft actor-critic. *in preparation*, 2019.

Hasselt, H. V. Double Q-learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2613–2621, 2010.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *Conference on Learning Representations (ICLR)*, 2018.

Heess, N., Silver, D., and Teh, Y. W. Actor-critic reinforcement learning with energy-based policies. In *European Workshop on Reinforcement Learning*, pp. 45–58, 2013.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2944–2952, 2015.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Conference on Artificial Intelligence (AAAI)*. AAAI, 2018.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems (NeurIPS)*, pp. 1547–1554, 2003.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2): 328–373, 2013.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*, 2017.

James, S., Davison, A. J., and Johns, E. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning (CoRL)*, pp. 334–343, 2017.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Kakade, S. M. A natural policy gradient. In *Advances in neural information processing systems (NeurIPS)*, pp. 1531–1538, 2002.

Kappen, H. J. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory And Experiment*, 2005(11):P11011, 2005.

Kenneally, G. D., De, A., and Koditschek, D. E. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1(2):900–907, 2016.

Kim, T. and Bengio, Y. Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*, 2016.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference for Learning Presentations (ICLR)*, 2015.

Kohl, N. and Stone, P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation*, volume 3, pp. 2619–2624. IEEE, 2004.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3675–3683, 2016.

Kupcsik, A. G., Deisenroth, M. P., Peters, J., and Neumann, G. Data-efficient generalization of robot skills with contextual policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1401–1407, 2013.

Lai, T. L. and Robbins, H. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436, 2015.

Levine, S. *Motor skill learning with local trajectory methods*. PhD thesis, Stanford University, 2014.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, Q. and Wang, D. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances In Neural Information Processing Systems (NeurIPS)*, pp. 2370–2378, 2016.

Liu, Y., Ramachandran, P., Liu, Q., and Peng, J. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399*, 2017.

MacAlpine, P. and Stone, P. Overlapping layered learning. *Artificial Intelligence*, 254:21–43, 2018.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning (CoRL)*, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2772–2782, 2017.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Trust-PCL: An off-policy trust region method for continuous control. In *International Conference on Learning Representations (ICLR)*, 2018.

Neumann, G. Variational inference for policy search in changing situations. In *International Conference on Machine Learning (ICML)*, pp. 817–824, 2011.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. Combining policy gradient and Q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

Otsuka, M., Yoshimoto, J., and Doya, K. Free-energy-based reinforcement learning in a partially observable environment. In *European Symposium on Artificial Neural Networks (ESANN)*, 2010.

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation (ICRA)*, pp. 763–768. IEEE, 2009.

Peng, X. B., Berseth, G., and Van de Panne, M. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

Peters, J. and Schaal, S. Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 2219–2225. IEEE, 2006.

Peters, J. and Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Peters, J., Mülling, K., and Altun, Y. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1607–1612, 2010.

Rai, A., Antonova, R., Song, S., Martin, W., Geyer, H., and Atkeson, C. G. Bayesian optimization using domain knowledge on the ATRIAS biped. *arXiv preprint arXiv:1709.06047*, 2017.

Rawlik, K., Toussaint, M., and Vijayakumar, S. On stochastic optimal control and reinforcement learning by approximate inference. In *Robotics: Science and Systems (RSS)*, pp. 3052–3056, 2012.

Sadeghi, F. and Levine, S. (CAD)^{2}RL: Real single-image flight without a single real image. In *Robotics: Science and Systems (RSS)*, 2017.

Sallans, B. and Hinton, G. E. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5(Aug):1063–1088, 2004.

Saxe, A. M., Earle, A. C., and Rosman, B. S. Hierarchy through composition with multitask LMDPs. *Proceedings of Machine Learning Research*, 2017.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning (ICML)*, pp. 1312–1320, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897, 2015a.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

Schulman, J., Abbeel, P., and Chen, X. Equivalence between policy gradients and soft Q-learning. *arXiv preprint arXiv:1704.06440*, 2017a.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017b.

Shelhamer, E., Mahmoudieh, P., Argus, M., and Darrell, T. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 0028-0836. Article.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems (RSS)*, 2018.

Tedrake, R., Zhang, T. W., and Seung, H. S. Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, volume 95585, pp. 1939–1412. Yale University New Haven (CT), 2005.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 3, pp. 6, 2017.

Theodorou, E., Buchli, J., and Schaal, S. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *International Conference on Robotics and Automation (ICRA)*, pp. 2397–2403. IEEE, 2010.

Thomas, P. Bias in natural actor-critic algorithms. In *International Conference on Machine Learning (ICML)*, pp. 441–448, 2014.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30. IEEE, 2017.

Todorov, E. Linearly-solvable Markov decision problems. In *Advances in neural information processing systems (NeurIPS)*, pp. 1369–1376, 2007.

Todorov, E. General duality between optimal control and estimation. In *Conference on Decision and Control (CDC)*, pp. 4286–4292. IEEE, 2008.

Todorov, E. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1856–1864, 2009.

Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033. IEEE, 2012.

Toussaint, M. Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)*, pp. 1049–1056. ACM, 2009.

Tung, H.-Y. F., Harley, A. W., Huang, L.-K., and Fragkiadaki, K. Reward learning from narrated demonstrations. *arXiv preprint arXiv:1804.10692*, 2018.

Uhlenbeck, G. E. and Ornstein, L. S. On the theory of the Brownian motion. *Physical review*, 36(5):823, 1930.

Večerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 3540–3549, 2017.

Wang, D. and Liu, Q. Learning to draw samples: With application to amortized MLE for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

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

Xie, Z., Berseth, G., Clary, P., Hurst, J., and van de Panne, M. Feedback control for cassie with deep reinforcement learning. *arXiv preprint arXiv:1803.05580*, 2018.

Zhang, F., Leitner, J., Milford, M., Upcroft, B., and Corke, P. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015.

Zhao, J., Mathieu, M., and LeCun, Y. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

Ziebart, B. D. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1433–1438, 2008.

Appendix A

Discounted Infinite Horizon Maximum Entropy Objective

The exact definition of the discounted maximum entropy objective is complicated by the fact that, when using a discount factor for policy gradient methods, we typically do not discount the state distribution, only the rewards. In that sense, discounted policy gradients typically do not optimize the true discounted objective. Instead, they optimize average reward, with the discount serving to reduce variance, as discussed by Thomas (2014). However, we can define the objective that is optimized under a discount factor as

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\rho_\pi} \left[\sum_{l=t}^{\infty} \gamma^{l-t} \, \mathbb{E}_{\mathbf{s}_l \sim p, \mathbf{a}_l \sim \pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t)) | \mathbf{s}_t, \mathbf{a}_t \right] \right]. \tag{A.1}$$

This objective corresponds to maximizing the discounted expected reward and entropy for future states originating from every state-action tuple $(\mathbf{s}_t, \mathbf{a}_t)$ weighted by its probability ρ_π under the current policy.

Appendix B

Proofs

B.1 Soft Q-Learning

The Maximum Entropy Policy

We start with the definition of the soft Q-value Q^π for any policy π as the expectation under π of the discounted sum of rewards and entropy:

$$Q^\pi(\mathbf{s}, \mathbf{a}) \triangleq r(\mathbf{s}_0, \mathbf{a}_0) + \mathbb{E}_{\tau\sim\pi, \mathbf{s}_0=\mathbf{s}, \mathbf{a}_0=\mathbf{a}}\left[\sum_{t=1}^{\infty} \gamma^t (r(\mathbf{s}_t, \mathbf{a}_t) + \mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t)))\right]. \tag{B.1}$$

Here, $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots)$ denotes the trajectory originating from $(\mathbf{s}, \mathbf{a})$. We have set $\alpha = 1$ for simplicity, but the theory can be easily adapted to other temperatures by dividing rewards by α. The discounted maximum entropy policy objective in Equation A.1 can now be defined as

$$J(\pi) \triangleq \sum_{t=0}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\rho_\pi}\left[Q^\pi(\mathbf{s}_t, \mathbf{a}_t) + \alpha\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t))\right]. \tag{B.2}$$

If the objective function is the expected discounted sum of rewards, the policy improvement theorem (Sutton & Barto, 1998) describes how policies can be improved monotonically. There is a similar theorem we can derive for the maximum entropy objective:

Theorem 4. *(Policy improvement theorem) Given a policy π, define a new policy $\tilde{\pi}$ as*

$$\tilde{\pi}(\,\cdot\,|\mathbf{s}) \propto \exp\left(Q^\pi(\mathbf{s}, \,\cdot\,)\right), \quad \forall \mathbf{s}. \tag{B.3}$$

Assume that throughout our computation, Q is bounded and $\int \exp(Q(\mathbf{s}, \mathbf{a}))\, d\mathbf{a}$ is bounded for any $\mathbf{s}$ (for both π and $\tilde{\pi}$). Then $Q^{\tilde{\pi}}(\mathbf{s}, \mathbf{a}) \geq Q^\pi(\mathbf{s}, \mathbf{a})$ $\forall \mathbf{s}, \mathbf{a}$.

The proof relies on the following observation: if one greedily maximize the sum of entropy and value with one-step look-ahead, then one obtains $\tilde{\pi}$ from π:

$$\mathcal{H}(\pi(\,\cdot\,|\mathbf{s})) + \mathbb{E}_{\mathbf{a}\sim\pi}\left[Q^\pi(\mathbf{s}, \mathbf{a})\right] \leq \mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s})) + \mathbb{E}_{\mathbf{a}\sim\tilde{\pi}}\left[Q^\pi(\mathbf{s}, \mathbf{a})\right]. \tag{B.4}$$

The proof is straight-forward by noticing that

$$\mathcal{H}(\pi(\,\cdot\,|\mathbf{s})) + \mathbb{E}_{\mathbf{a}\sim\pi}\left[Q^\pi(\mathbf{s},\mathbf{a})\right] = -\mathrm{D}_{\mathrm{KL}}\left(\pi(\,\cdot\,|\mathbf{s}) \;\|\; \tilde{\pi}(\,\cdot\,|\mathbf{s})\right) + \log\int \exp\left(Q^\pi(\mathbf{s},\mathbf{a})\right)\, d\mathbf{a} \tag{B.5}$$

Then we can show that

$$\begin{aligned}
Q^\pi(\mathbf{s},\mathbf{a}) &= \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma(\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_1)) + \mathbb{E}_{\mathbf{a}_1\sim\pi}\left[Q^\pi(\mathbf{s}_1,\mathbf{a}_1)\right])\right] \\
&\le \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma(\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_1)) + \mathbb{E}_{\mathbf{a}_1\sim\tilde{\pi}}\left[Q^\pi(\mathbf{s}_1,\mathbf{a}_1)\right])\right] \\
&= \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma(\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_1)) + r_1)\right] + \gamma^2\,\mathbb{E}_{\mathbf{s}_2}\left[\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_2)) + \mathbb{E}_{\mathbf{a}_2\sim\pi}\left[Q^\pi(\mathbf{s}_2,\mathbf{a}_2)\right]\right] \\
&\le \mathbb{E}_{\mathbf{s}_1}\left[r_0 + \gamma(\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_1)) + r_1\right] + \gamma^2\,\mathbb{E}_{\mathbf{s}_2}\left[\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_2)) + \mathbb{E}_{\mathbf{a}_2\sim\tilde{\pi}}\left[Q^\pi(\mathbf{s}_2,\mathbf{a}_2)\right]\right] \\
&= \mathbb{E}_{\mathbf{s}_1\,\mathbf{a}_2\sim\tilde{\pi},\mathbf{s}_2}\left[r_0 + \gamma(\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_1)) + r_1) + \gamma^2(\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_2)) + r_2)\right] \\
&\quad + \gamma^3\,\mathbb{E}_{\mathbf{s}_3}\left[\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_3)) + \mathbb{E}_{\mathbf{a}_3\sim\tilde{\pi}}\left[Q^\pi(\mathbf{s}_3,\mathbf{a}_3)\right]\right] \\
&\;\;\vdots \\
&\le \mathbb{E}_{\tau\sim\tilde{\pi}}\left[r_0 + \sum_{t=1}^{\infty}\gamma^t(\mathcal{H}(\tilde{\pi}(\,\cdot\,|\mathbf{s}_t)) + r_t)\right] \\
&= Q^{\tilde{\pi}}(\mathbf{s},\mathbf{a}).
\end{aligned} \tag{B.6}$$

With Theorem 4, we start from an arbitrary policy π_0 and define the *policy iteration* as

$$\pi_{i+1}(\,\cdot\,|\mathbf{s}) \propto \exp\left(Q^{\pi_i}(\mathbf{s},\,\cdot\,)\right). \tag{B.7}$$

Then $Q^{\pi_i}(\mathbf{s},\mathbf{a})$ improves monotonically. Under certain regularity conditions, π_i converges to π_∞. Obviously, we have $\pi_\infty(\mathbf{a}|\mathbf{s}) \propto_{\mathbf{a}} \exp\left(Q^{\pi_\infty}(\mathbf{s},\mathbf{a})\right)$. Since any non-optimal policy can be improved this way, the optimal policy must satisfy this energy-based form. Therefore we have proven Lemma 1.

Soft Bellman Equation and Soft Value Iteration

Recall the definition of the soft value function:

$$V^\pi(\mathbf{s}) \triangleq \log\int \exp\left(Q^\pi(\mathbf{s},\mathbf{a})\right)\, d\mathbf{a}. \tag{B.8}$$

Suppose $\pi(\mathbf{a}|\mathbf{s}) = \exp\left(Q^\pi(\mathbf{s},\mathbf{a}) - V^\pi(\mathbf{s})\right)$. Then we can show that

$$\begin{aligned}
Q^\pi(\mathbf{s},\mathbf{a}) &= r(\mathbf{s},\mathbf{a}) + \gamma\,\mathbb{E}_{\mathbf{s}'\sim p}\left[\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}')) + \mathbb{E}_{\mathbf{a}'\sim\pi(\,\cdot\,|\mathbf{s}')}\left[Q^\pi(\mathbf{s}',\mathbf{a}')\right]\right] \\
&= r(\mathbf{s},\mathbf{a}) + \gamma\,\mathbb{E}_{\mathbf{s}'\sim p}\left[V^\pi(\mathbf{s}')\right].
\end{aligned} \tag{B.9}$$

This completes the proof of Lemma 2.

Finally, we show that the soft value iteration operator $\mathcal{T}$, defined as

$$\mathcal{T}Q(\mathbf{s},\mathbf{a}) \triangleq r(\mathbf{s},\mathbf{a}) + \gamma\,\mathbb{E}_{\mathbf{s}'\sim p}\left[\log\int \exp Q(\mathbf{s}',\mathbf{a}')\; d\mathbf{a}'\right], \tag{B.10}$$

is a contraction. Then Theorem 1 follows immediately. The following proof has also been presented by Fox et al. (2016). Define a norm on Q-values as $\|Q_1 - Q_2\| \triangleq \max_{\mathbf{s},\mathbf{a}} |Q_1(\mathbf{s},\mathbf{a}) - Q_2(\mathbf{s},\mathbf{a})|$. Suppose $\varepsilon = \|Q_1 - Q_2\|$. Then

$$\begin{aligned}\log \int \exp(Q_1(\mathbf{s}', \mathbf{a}'))\, d\mathbf{a}' &\le \log \int \exp(Q_2(\mathbf{s}', \mathbf{a}') + \varepsilon)\, d\mathbf{a}' \\ &= \log \left(\exp(\varepsilon) \int \exp Q_2(\mathbf{s}', \mathbf{a}')\, d\mathbf{a}' \right) \\ &= \varepsilon + \log \int \exp Q_2(\mathbf{a}', \mathbf{a}')\, d\mathbf{a}'. \end{aligned} \tag{B.11}$$

Similarly, $\log \int \exp Q_1(\mathbf{s}', \mathbf{a}')\, d\mathbf{a}' \ge -\varepsilon + \log \int \exp Q_2(\mathbf{s}', \mathbf{a}')\, d\mathbf{a}'$. Therefore $\|\mathcal{T}Q_1 - \mathcal{T}Q_2\| \le \gamma\varepsilon = \gamma\|Q_1 - Q_2\|$. So $\mathcal{T}$ is a contraction. As a consequence, only one Q-value satisfies the soft Bellman equation, and thus the optimal policy presented in Lemma 1 is unique.

B.2 Connection between Policy Gradient and Q-Learning

We show that entropy-regularized policy gradient can be viewed as performing soft Q-learning on the maximum-entropy objective. First, suppose that we parametrize a stochastic policy as

$$\pi^\phi(\mathbf{a}_t|\mathbf{s}_t) \triangleq \exp\left(\mathcal{E}^\phi(\mathbf{s}_t, \mathbf{a}_t) - \bar{\mathcal{E}}^\phi(\mathbf{s}_t)\right), \tag{B.12}$$

where $\mathcal{E}^\phi(\mathbf{s}_t, \mathbf{a}_t)$ is an energy function with parameters ϕ, and $\bar{\mathcal{E}}^\phi(\mathbf{s}_t) = \log \int_{\mathcal{A}} \exp \mathcal{E}^\phi(\mathbf{s}_t, \mathbf{a}_t) d\mathbf{a}_t$ is the corresponding partition function. This is the most general class of policies, as we can trivially transform any given distribution p into exponential form by defining the energy as $\log p$. We can write an entropy-regularized policy gradient as follows:

$$\nabla_\phi J_{\mathrm{PG}}(\phi) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_{\pi^\phi}}\left[\nabla_\phi \log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t)\left(\hat{Q}_{\pi^\phi}(\mathbf{s}_t, \mathbf{a}_t) + b^\phi(\mathbf{s}_t)\right)\right] + \nabla_\phi \mathbb{E}_{\mathbf{s}_t\sim\rho_{\pi^\phi}}\left[\mathcal{H}(\pi^\phi(\,\cdot\,|\mathbf{s}_t))\right], \tag{B.13}$$

where $\rho_{\pi^\phi}(\mathbf{s}_t, \mathbf{a}_t)$ is the distribution induced by the policy, $\hat{Q}_{\pi^\phi}(\mathbf{s}_t, \mathbf{a}_t)$ is an empirical estimate of the Q-value of the policy, and $b^\phi(\mathbf{s}_t)$ is a state-dependent baseline that we get to choose. The gradient of the entropy term is given by

$$\begin{aligned}\nabla_\phi \mathcal{H}(\pi^\phi) &= -\nabla_\phi \mathbb{E}_{\mathbf{s}_t\sim\rho_{\pi^\phi}}\left[\mathbb{E}_{\mathbf{a}_t\sim\pi^\phi(\mathbf{a}_t|\mathbf{s}_t)}\left[\log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t)\right]\right] \\ &= -\mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_{\pi^\phi}}\left[\nabla_\phi \log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t) \log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t) + \nabla_\phi \log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t)\right] \\ &= -\mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_{\pi^\phi}}\left[\nabla_\phi \log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t)\left(1 + \log \pi^\phi(\mathbf{a}_t|\mathbf{s}_t)\right)\right],\end{aligned} \tag{B.14}$$

and after substituting this back into Equation B.13, noting Equation B.12, and choosing $b^\phi(\mathbf{s}_t) = \bar{\mathcal{E}}^\phi(\mathbf{s}_t) + 1$, we arrive at a simple form for the policy gradient:

$$= \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_{\pi^\phi}}\left[\left(\nabla_\phi \mathcal{E}^\phi(\mathbf{s}_t,\mathbf{a}_t) - \nabla_\phi \bar{\mathcal{E}}^\phi(\mathbf{s}_t)\right)\left(\hat{Q}_{\pi^\phi}(\mathbf{s}_t,\mathbf{a}_t) - \mathcal{E}^\phi(\mathbf{s}_t,\mathbf{a}_t)\right)\right]. \tag{B.15}$$

To show that Equation B.15 indeed corresponds to soft Q-learning update, we consider the Bellman error

$$J_Q(\theta) = \mathbb{E}_{\mathbf{s}_t\sim q_{\mathbf{s}_t},\mathbf{a}_t\sim q_{\mathbf{a}_t}}\left[\frac{1}{2}\left(\hat{Q}^\theta(\mathbf{s}_t,\mathbf{a}_t) - Q^\theta(\mathbf{s}_t,\mathbf{a}_t)\right)^2\right], \tag{B.16}$$

where $\hat{Q}^\theta$ is an empirical estimate of the soft Q-function. There are several valid alternatives for this estimate, but in order to show a connection to policy gradient, we choose a specific form

$$\hat{Q}^\theta(\mathbf{s}_t,\mathbf{a}_t) = \hat{A}^{\bar{\theta}}(\mathbf{s}_t,\mathbf{a}_t) + V^\theta(\mathbf{s}_t), \tag{B.17}$$

where $\hat{A}^{\bar{\theta}}$ is an empirical soft advantage function that is assumed not to contribute the gradient computation. With this choice, the gradient of the Bellman error becomes

$$\begin{aligned}\nabla_\theta J_Q(\theta) &= \mathbb{E}_{\mathbf{s}_t\sim q_{\mathbf{s}_t},\mathbf{a}_t\sim q_{\mathbf{a}_t}}\left[\left(\nabla_\theta Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \nabla_\theta V_\theta(\mathbf{s}_t)\right)\left(\hat{A}^{\bar{\theta}}(\mathbf{s}_t,\mathbf{a}_t) + V^\theta(\mathbf{s}_t,\mathbf{a}_t) - Q_\theta(\mathbf{s}_t,\mathbf{a}_t)\right)\right]\\ &= \mathbb{E}_{\mathbf{s}_t\sim q_{\mathbf{s}_t},\mathbf{a}_t\sim q_{\mathbf{a}_t}}\left[\left(\nabla_\theta Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \nabla_\theta V_\theta(\mathbf{s}_t)\right)\left(\hat{Q}^\theta(\mathbf{s}_t,\mathbf{a}_t) - Q^\theta(\mathbf{s}_t,\mathbf{a}_t)\right)\right].\end{aligned} \tag{B.18}$$

Now, if we choose $\mathcal{E}^\phi(\mathbf{s}_t,\mathbf{a}_t) \triangleq Q_\theta(\mathbf{s}_t,\mathbf{a}_t)$ and $q_{\mathbf{s}_t}(\mathbf{s}_t)q_{\mathbf{a}_t}(\mathbf{a}_t) \triangleq \rho_{\pi^\phi}(\mathbf{s}_t,\mathbf{a}_t)$, we recover the policy gradient in Equation B.15. Note that the choice of using an empirical estimate of the soft advantage rather than soft Q-value makes the target independent of the soft value, and at convergence, Q_θ approximates the soft Q-value up to an additive constant. The resulting policy is still correct, since the Boltzmann distribution in Equation B.12 is independent of constant shift in the energy function.

B.3 Soft Actor-Critic

Lemma 3 (Soft Policy Evaluation). *Consider the soft Bellman backup operator $\mathcal{T}^\pi$ in Equation 4.1 and a mapping $Q^0 : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ with $|\mathcal{A}|<\infty$, and define $Q^{k+1} = \mathcal{T}^\pi Q^k$. Then the sequence Q^k will converge to the soft Q-value of π as $k\to\infty$.*

Proof. Define the entropy augmented reward as $r_\pi(\mathbf{s}_t,\mathbf{a}_t) \triangleq r(\mathbf{s}_t,\mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[\mathcal{H}\left(\pi(\,\cdot\,|\mathbf{s}_{t+1})\right)\right]$ and rewrite the update rule as

$$Q(\mathbf{s}_t,\mathbf{a}_t) \leftarrow r_\pi(\mathbf{s}_t,\mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s}_{t+1}\sim p,\mathbf{a}_{t+1}\sim\pi}\left[Q(\mathbf{s}_{t+1},\mathbf{a}_{t+1})\right] \tag{B.19}$$

and apply the standard convergence results for policy evaluation (Sutton & Barto, 1998). The assumption $|\mathcal{A}|<\infty$ is required to guarantee that the entropy augmented reward is bounded. □

Lemma 4 (Soft Policy Improvement). *Let $\pi_{\text{old}} \in \Pi$ and let π_{new} be the optimizer of the minimization problem defined in Equation 4.3. Then $Q^{\pi_{\text{new}}}(\mathbf{s}_t, \mathbf{a}_t) \geq Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t)$ for all $(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{S} \times \mathcal{A}$ with $|\mathcal{A}| < \infty$.*

Proof. Let $\pi_{\text{old}} \in \Pi$ and let $Q^{\pi_{\text{old}}}$ and $V^{\pi_{\text{old}}}$ be the corresponding soft state-action value and soft state value, and let π_{new} be defined as

$$\begin{aligned}\pi_{\text{new}}(\,\cdot\,|\mathbf{s}_t) &= \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\,\cdot\,|\mathbf{s}_t) \,\|\, \exp\left(Q^{\pi_{\text{old}}}(\mathbf{s}_t, \,\cdot\,) - \log Z^{\pi_{\text{old}}}(\mathbf{s}_t)\right)\right) \\ &= \arg\min_{\pi' \in \Pi} J_{\pi_{\text{old}}}(\pi'(\,\cdot\,|\mathbf{s}_t)). \end{aligned} \tag{B.20}$$

It must be the case that $J_{\pi_{\text{old}}}(\pi_{\text{new}}(\,\cdot\,|\mathbf{s}_t)) \leq J_{\pi_{\text{old}}}(\pi_{\text{old}}(\,\cdot\,|\mathbf{s}_t))$, since we can always choose $\pi_{\text{new}} = \pi_{\text{old}} \in \Pi$. Hence

$$\begin{aligned}&\mathbb{E}_{\mathbf{a}_t \sim \pi_{\text{new}}}\left[\log \pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t) - Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t) + \log Z^{\pi_{\text{old}}}(\mathbf{s}_t)\right] \\ &\quad \leq \mathbb{E}_{\mathbf{a}_t \sim \pi_{\text{old}}}\left[\log \pi_{\text{old}}(\mathbf{a}_t|\mathbf{s}_t) - Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t) + \log Z^{\pi_{\text{old}}}(\mathbf{s}_t)\right], \end{aligned} \tag{B.21}$$

and since partition function $Z^{\pi_{\text{old}}}$ depends only on the state, the inequality reduces to

$$\mathbb{E}_{\mathbf{a}_t \sim \pi_{\text{new}}}\left[Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t)\right] \geq V^{\pi_{\text{old}}}(\mathbf{s}_t). \tag{B.22}$$

Next, consider the soft Bellman equation:

$$\begin{aligned} Q^{\pi_{\text{old}}}(\mathbf{s}_t, \mathbf{a}_t) &= r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1} \sim p}\left[V^{\pi_{\text{old}}}(\mathbf{s}_{t+1})\right] \\ &\leq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1} \sim p}\left[\mathbb{E}_{\mathbf{a}_{t+1} \sim \pi_{\text{new}}}\left[Q^{\pi_{\text{old}}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \log \pi_{\text{new}}(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})\right]\right] \\ &\;\;\vdots \\ &\leq Q^{\pi_{\text{new}}}(\mathbf{s}_t, \mathbf{a}_t), \end{aligned} \tag{B.23}$$

where we have repeatedly expanded $Q^{\pi_{\text{old}}}$ on the RHS by applying the soft Bellman equation and the bound in Equation B.22. Convergence to $Q^{\pi_{\text{new}}}$ follows from Lemma 3. □

Theorem 2 (Soft Policy Iteration). *Repeated application of soft policy evaluation and soft policy improvement to any $\pi \in \Pi$ converges to a policy π^* such that $Q^{\pi^*}(\mathbf{s}_t, \mathbf{a}_t) \geq Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ for all $\pi \in \Pi$ and $(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{S} \times \mathcal{A}$, assuming $|\mathcal{A}| < \infty$.*

Proof. Let π_i be the policy at iteration i. By Lemma 4, the sequence Q^{π_i} is monotonically increasing. Since Q^{π} is bounded above for $\pi \in \Pi$ (both the reward and entropy are bounded), the sequence converges to some π^*. We will still need to show that π^* is indeed optimal. At convergence, it must be case that $J_{\pi^*}(\pi^*(\,\cdot\,|\mathbf{s}_t)) < J_{\pi^*}(\pi(\,\cdot\,|\mathbf{s}_t))$ for all $\pi \in \Pi$, $\pi \neq \pi^*$. Using the same iterative argument as in the proof of Lemma 4, we get $Q^{\pi^*}(\mathbf{s}_t, \mathbf{a}_t) > Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ for all $(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{S} \times \mathcal{A}$, that is, the soft value of any other policy in Π is lower than that of the converged policy. Hence π^* is optimal in Π. □

B.4 Policy Composition Proofs

Lemma 5: *Let Q_1^* and Q_2^* be the soft Q-function of the optimal policies corresponding to reward functions r_1 and r_2, and define $Q_\Sigma \triangleq \frac{1}{2}(Q_1^* + Q_2^*)$. Then the optimal soft Q-function of the combined reward $r_\mathcal{C} \triangleq \frac{1}{2}(r_1 + r_2)$ satisfies*

$$Q_\Sigma(\mathbf{s}, \mathbf{a}) \geq Q_\mathcal{C}^*(\mathbf{s}, \mathbf{a}) \geq Q_\Sigma(\mathbf{s}, \mathbf{a}) - C^*(\mathbf{s}, \mathbf{a}),\ \forall \mathbf{s} \in \mathcal{S}, \forall \mathbf{a} \in \mathcal{A}, \tag{B.24}$$

where C^* is the fixed point of

$$C(\mathbf{s}, \mathbf{a}) \leftarrow \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathcal{D}_{\frac{1}{2}} \left(\pi_1^*(\,\cdot\,|\mathbf{s}') \parallel \pi_2^*(\,\cdot\,|\mathbf{s}') \right) + \max_{\mathbf{a}' \in \mathcal{A}} C(\mathbf{s}', \mathbf{a}') \right], \tag{B.25}$$

and $\mathcal{D}_{\frac{1}{2}}(\,\cdot \parallel \cdot\,)$ is the Rényi divergence of order $1/2$.

Proof. We will first prove the lower bound using induction. Let us define functions $Q^{(0)}(\mathbf{s}, \mathbf{a}) \triangleq Q_\Sigma(\mathbf{s}, \mathbf{a})$ and $C^{(0)}(\mathbf{s}, \mathbf{a}) \triangleq 0$, and assume $Q^{(k)} \geq Q_\Sigma - C^{(k)}$ for some k. Note that this inequality is trivially satisfied when $k = 0$. Next, let us apply the soft Bellman backup at step k:

$$\begin{aligned}
&Q^{(k+1)}(\mathbf{s}, \mathbf{a}) \\
&\quad = r_\mathcal{C}(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\log \int_\mathcal{A} \exp Q^{(k)}(\mathbf{s}', \mathbf{a}') d\mathbf{a}' \right] \\
&\quad \geq r_\mathcal{C}(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\log \int_\mathcal{A} \exp \left(\frac{1}{2} \left(Q_1^*(\mathbf{s}', \mathbf{a}') + Q_2^*(\mathbf{s}', \mathbf{a}') \right) - C^{(k)}(\mathbf{s}', \mathbf{a}') \right) d\mathbf{a}' \right] \\
&\quad \geq r_\mathcal{C}(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\log \int_\mathcal{A} \exp \frac{1}{2} \left(Q_1^*(\mathbf{s}', \mathbf{a}') + Q_2^*(\mathbf{s}', \mathbf{a}') \right) d\mathbf{a}' - \max_{\mathbf{a}'} C^{(k)}(\mathbf{s}', \mathbf{a}') \right] \\
&\quad = r_\mathcal{C}(\mathbf{s}, \mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\frac{1}{2} \left(V_1^*(\mathbf{s}') + V_2^*(\mathbf{s}') \right) + \log \int_\mathcal{A} \sqrt{\pi_1^*(\mathbf{a}'|\mathbf{s}')\pi_2^*(\mathbf{a}'|\mathbf{s}')} d\mathbf{a}' - \max_{\mathbf{a}'} C^{(k)}(\mathbf{s}', \mathbf{a}') \right] \\
&\quad = \frac{1}{2} \left(Q_1^*(\mathbf{s}, \mathbf{a}) + Q_2^*(\mathbf{s}, \mathbf{a}) \right) + \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\log \int_\mathcal{A} \sqrt{\pi_1^*(\mathbf{a}'|\mathbf{s}')\pi_2^*(\mathbf{a}'|\mathbf{s}')} d\mathbf{a}' - \max_{\mathbf{a}'} C^{(k)}(\mathbf{s}', \mathbf{a}') \right] \\
&\quad = Q_\Sigma(\mathbf{s}, \mathbf{a}) - \gamma\, \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\frac{1}{2} \mathcal{D}_{\frac{1}{2}} \left(\pi_1^*(\,\cdot\,|\mathbf{s}') \parallel \pi_2^*(\,\cdot\,|\mathbf{s}') \right) + \max_{\mathbf{a}'} C^{(k)}(\mathbf{s}', \mathbf{a}') \right] \\
&\quad = Q_\Sigma(\mathbf{s}, \mathbf{a}) - C^{(k+1)}(\mathbf{s}, \mathbf{a}).
\end{aligned} \tag{B.26}$$

The last form shows that the induction hypothesis is true for $k + 1$. Since soft Bellman iteration converges to the optimal soft Q-function from any bounded $Q^{(0)}$, at the limit, we will have

$$Q_\mathcal{C}^*(\mathbf{s}, \mathbf{a}) \geq Q_\Sigma(\mathbf{s}, \mathbf{a}) - C^*(\mathbf{s}, \mathbf{a}), \tag{B.27}$$

where the value of C^* is the fixed point of the following recursion:

$$C^{(k+1)} = \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\frac{1}{2}\mathcal{D}_{\frac{1}{2}} \left(\pi_1^*(\,\cdot\,|\mathbf{s}') \parallel \pi_2^*(\,\cdot\,|\mathbf{s}')\right) + \max_{\mathbf{a}'} C^{(k)}(\mathbf{s}',\mathbf{a}') \right]. \tag{B.28}$$

The upper bound follows analogously by assuming $Q^{(k)} \leq Q_\Sigma$, defining $Q^{(0)} = Q_\Sigma$ and noting that the Rényi divergence is always positive. $\square$

Corollary 1. *As a corollary of Lemma 5, we have*

$$V_\Sigma(\mathbf{s}) \geq V_\mathcal{C}^*(\mathbf{s}) \geq V_\Sigma(\mathbf{s}) - \max_{\mathbf{a}} C^*(\mathbf{s},\mathbf{a}),\ \forall \mathbf{s}\in\mathcal{S}, \tag{B.29}$$

where $V_\Sigma(\mathbf{s}) = \log\int_\mathcal{A} \exp(Q(\mathbf{s},\mathbf{a}))d\mathbf{a}$

Proof. Take the "log-sum-exp" of (B.24). $\square$

Proof of Theorem 3

Theorem 3: *With the definitions in Lemma 5, the value of* π_Σ *satisfies*

$$Q_\mathcal{C}^{\pi_\Sigma}(\mathbf{s},\mathbf{a}) \geq Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) - D^*(\mathbf{s},\mathbf{a}), \tag{B.30}$$

where $D^*(\mathbf{s},\mathbf{a})$ *is the fixed point of*

$$D(\mathbf{s},\mathbf{a}) \leftarrow \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathbb{E}_{\mathbf{a}'\sim\pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[C^*(\mathbf{s}',\mathbf{a}') + D(\mathbf{s}',\mathbf{a}') \right]\right]. \tag{B.31}$$

Proof. Let us define functions $Q^{(0)}(\mathbf{s},\mathbf{a}) \triangleq Q_\mathcal{C}^*(\mathbf{s},\mathbf{a})$ and $D^{(0)}(\mathbf{s},\mathbf{a}) \triangleq 0$, and assume that $Q^{(k)}(\mathbf{s},\mathbf{a}) \geq Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) - D^{(k)}(\mathbf{s},\mathbf{a})$, which is trivially satisfied for $k = 0$. We will use induction to show the inequality holds for any k by applying "soft policy evaluation" (see (Haarnoja et al., 2017)):

$$\begin{aligned}
&Q^{(k+1)}(\mathbf{s},\mathbf{a})\\
&\quad= r_\mathcal{C}(\mathbf{s},\mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathbb{E}_{\mathbf{a}'\sim\pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[Q^{(k)}(\mathbf{s}',\mathbf{a}') - \log\pi_\Sigma(\mathbf{a}'|\mathbf{s}') \right]\right]\\
&\quad\geq r_\mathcal{C}(\mathbf{s},\mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathbb{E}_{\mathbf{a}'\sim\pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[Q_\mathcal{C}^*(\mathbf{s}',\mathbf{a}') - D^{(k)}(\mathbf{s}',\mathbf{a}') - Q_\Sigma(\mathbf{s}',\mathbf{a}') + V_\Sigma(\mathbf{s}') \right]\right]\\
&\quad\geq r_\mathcal{C}(\mathbf{s},\mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathbb{E}_{\mathbf{a}'\sim\pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[Q_\mathcal{C}^*(\mathbf{s}',\mathbf{a}') - D^{(k)}(\mathbf{s}',\mathbf{a}') - Q_\mathcal{C}^*(\mathbf{s}',\mathbf{a}') - C^*(\mathbf{s}',\mathbf{a}') + V_\mathcal{C}^*(\mathbf{s}') \right]\right]\\
&\quad\geq r_\mathcal{C}(\mathbf{s},\mathbf{a}) + \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[V_\mathcal{C}^*(\mathbf{s}') - \mathbb{E}_{\mathbf{a}'\sim\pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[C^*(\mathbf{s}',\mathbf{a}') + D^{(k)}(\mathbf{s}',\mathbf{a}') \right]\right]\\
&\quad= Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) - \gamma\, \mathbb{E}_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\mathbb{E}_{\mathbf{a}'\sim\pi_\Sigma(\mathbf{a}'|\mathbf{s}')} \left[C^*(\mathbf{s}',\mathbf{a}') + D^{(k)}(\mathbf{s}',\mathbf{a}') \right]\right]\\
&\quad= Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) - D^{(k+1)}(\mathbf{s},\mathbf{a}).
\end{aligned} \tag{B.32}$$

The second inequality follows from Lemma 5 and Corollary 1. In the limit as $k\to\infty$, we have $Q_\mathcal{C}^{\pi_\Sigma}(\mathbf{s},\mathbf{a}) \geq Q_\mathcal{C}^*(\mathbf{s},\mathbf{a}) - D^*(\mathbf{s},\mathbf{a})$. $\square$

Appendix C

Implementation

C.1 Soft Q-Learning

Computing Soft Q-Learning Policy Update

Here we explain in full detail how the policy update direction $\hat{\nabla}_\phi J_\pi$ in Algorithm 1 is computed. We reuse the indices i, j in this section with a different meaning than in the body of the paper for the sake of providing a cleaner presentation.

Expectations appear in amortized SVGD in two places. First, SVGD approximates the optimal descent direction $\phi(\,\cdot\,)$ in Equation (3.11) with an empirical average over the samples $\mathbf{a}_t^{(i)} = f^\phi(\xi^{(i)})$. Similarly, SVGD approximates the expectation in Equation (3.12) with samples $\tilde{\mathbf{a}}_t^{(j)} = f^\phi(\tilde{\xi}^{(j)})$, which can be the same or different from $\mathbf{a}_t^{(i)}$. Substituting (3.11) into (3.12) and taking the gradient gives the empirical estimate

$$\hat{\nabla}_\phi J_\pi(\phi; \mathbf{s}_t) = \frac{1}{KM} \sum_{j=1}^{K} \sum_{i=1}^{M} \left(\kappa(\mathbf{a}_t^{(i)}, \tilde{\mathbf{a}}_t^{(j)}) \nabla_{\mathbf{a}'} Q(\mathbf{s}_t, \mathbf{a}')\big|_{\mathbf{a}'=\mathbf{a}_t^{(i)}} + \nabla_{\mathbf{a}'} \kappa(\mathbf{a}', \tilde{\mathbf{a}}_t^{(j)})\big|_{\mathbf{a}'=\mathbf{a}_t^{(i)}} \right) \nabla_\phi f^\phi(\tilde{\xi}^{(j)}; \mathbf{s}_t).$$

Finally, the update direction $\hat{\nabla}_\phi J_\pi$ is the average of $\hat{\nabla}_\phi J_\pi(\phi; \mathbf{s}_t)$, where $\mathbf{s}_t$ is drawn from a mini-batch.

Hyperparameters

Throughout all experiments, we use the following parameters for both DDPG and soft Q-learning. The Q-values are updated using ADAM with learning rate 0.001. The DDPG policy and soft Q-learning sampling network use ADAM with a learning rate of 0.0001. The algorithm uses a replay pool of size one million. Training does not start until the replay pool has at least 10,000 samples. Every mini-batch has size 64. Each training iteration consists of 10000 time steps, and both the Q-values and policy / sampling network are trained at every time step. All experiments are run for 500 epochs, except that the multi-goal task uses 100 epochs and the fine-tuning tasks are trained for 200 epochs. Both the Q-value and policy

/ sampling network are neural networks comprised of two hidden layers, with 200 hidden units at each layer and ReLU nonlinearity. Both DDPG and soft Q-learning use additional OU Noise (Uhlenbeck & Ornstein, 1930; Lillicrap et al., 2015) to improve exploration. The parameters are $\theta = 0.15$ and $\sigma = 0.3$. In addition, we found that updating the target parameters too frequently can destabilize training. Therefore we freeze target parameters for every 1000 time steps (except for the swimming snake experiment, which freezes for 5000 epochs), and then copy the current network parameters to the target networks directly ($\tau = 1$).

Soft Q-learning uses $K = M = 32$ action samples (see Appendix C.1) to compute the policy update, except that the multi-goal experiment uses $K = M = 100$. The number of additional action samples to compute the soft value is $K_V = 50$. The kernel κ is a radial basis function, written as $\kappa(\mathbf{a}, \mathbf{a}') = \exp(-\frac{1}{h}\|\mathbf{a} - \mathbf{a}'\|_2^2)$, where $h = \frac{d}{2\log(M+1)}$, with d equal to the median of pairwise distance of sampled actions $\mathbf{a}_t^{(i)}$. Note that the step size h changes dynamically depending on the state $\mathbf{s}$, as suggested in (Liu & Wang, 2016).

The entropy coefficient α is 10 for multi-goal environment, and 0.1 for the swimming snake, maze, hallway (pretraining) and U-shaped maze (pretraining) experiments.

All fine-tuning tasks anneal the entropy coefficient α quickly in order to improve performance, since the goal during fine-tuning is to recover a near-deterministic policy on the fine-tuning task. In particular, α is annealed log-linearly to 0.001 within 20 epochs of fine-tuning. Moreover, the samples ξ are fixed to a set $\{\xi_i\}_{i=1}^{K_\xi}$ and K_ξ is reduced linearly to 1 within 20 epochs.

Task description

All tasks have a horizon of $T = 500$, except the multi-goal task, which uses $T = 20$. We add an additional termination condition to the quadrupedal 3D robot to discourage it from flipping over.

C.2 Soft Actor-Critic

Table C.1 lists the common SAC parameters used in the comparative evaluation in Figure 4.1 and Figure D.2. Table C.2 lists the reward scale parameter that was tuned for each environment.

Table C.1: SAC Hyperparameters

Parameter	Value
Shared	
optimizer	Adam (Kingma & Ba, 2015)
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	10^6
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per minibatch	256
nonlinearity	ReLU
SAC	
target smoothing coefficient (τ)	0.005
target update interval	1
gradient steps	1
SAC (hard target update)	
target smoothing coefficient (τ)	1
target update interval	1000
gradient steps (except humanoids)	4
gradient steps (humanoids)	1

Table C.2: SAC Environment Specific Parameters

Environment	Action Dimensions	Reward Scale
Hopper-v1	3	5
Walker2d-v1	6	5
HalfCheetah-v1	6	5
Ant-v1	8	5
Humanoid-v1	17	20
Humanoid (rllab)	21	10

C.3 Enforcing Action Bounds

In order to use a Gaussian policy, the actions needs to be bounded to a finite interval. To that end, we apply an invertible squashing function (tanh) to the Gaussian samples, and employ the change of variables formula to compute the likelihoods of the bounded actions. In the other words, let $\mathbf{u} \in \mathbb{R}^{\dim(\mathcal{A})}$ be a random variable and $\mu(\mathbf{u}|\mathbf{s})$ the corresponding density with infinite support. Then $\mathbf{a} = \tanh(\mathbf{u})$, where tanh is applied element-wise, is a random variable with support in $(-1, 1)$ with a density given by

$$\pi(\mathbf{a}|\mathbf{s}) = \mu(\mathbf{u}|\mathbf{s}) \left| \det \left(\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}\mathbf{u}} \right) \right|^{-1}. \tag{C.1}$$

Since the Jacobian ${}^{\mathrm{d}\mathbf{a}}/_{\mathrm{d}\mathbf{u}} = \mathrm{diag}(1 - \tanh^2(\mathbf{u}))$ is diagonal, the log-likelihood has a simple form

$$\log \pi(\mathbf{a}|\mathbf{s}) = \log \mu(\mathbf{u}|\mathbf{s}) - \sum_{i=1}^{\dim(\mathcal{A})} \log \left(1 - \tanh^2(u_i)\right), \tag{C.2}$$

where u_i is the i^{th} element of $\mathbf{u}$.

C.4 Latent Space Policies

Common Parameters

We use the following parameters for LSP policies throughout the experiments. The algorithm uses a replay pool of one million samples, and the training is delayed until at least 1000 samples have been collected to the pool. Each training iteration consists of 1000 environments time steps, and all the networks (value functions, policy scale/translation, and observation embedding network) are trained at every time step. Every training batch has a size of 128. The value function networks and the embedding network are all neural networks comprised of two hidden layers, with 128 ReLU units at each hidden layer. The dimension of the observation embedding is equal to two times the number of action dimensions. The scale and translation neural networks used in the real NVP bijector both have one hidden layer consisting of number of ReLU units equal to the number of action dimensions. All the network parameters are updated using Adam optimizer with learning rate $3 \cdot 10^{-4}$.

Table C.3 lists the common parameters used for the LSP-policy, and Table C.4 lists the parameters that varied across the environments.

High-Level Policies

All the low-level policies in hierarchical cases (Figures 7.5b, 7.4a, 7.4b) are trained using the same parameters used for the corresponding benchmark environment. All the high-level

Table C.3: Shared parameters for benchmark tasks

Parameter	Value
learning rate	$3 \cdot 10^{-4}$
batch size	128
discount	0.99
target smoothing coefficient	10^{-2}
maximum path length	10^{3}
replay pool size	10^{6}
hidden layers (Q, V, embedding)	2
hidden units per layer (Q, V, embedding)	128
policy coupling layers	2

Table C.4: Environment specific parameters for benchmark tasks

Parameter	Swimmer (rllab)	Hopper-v1	Walker2d-v1	HalfCheetah-v1	Ant (rllab)	Humanoid (rllab)
action dimensions	2	3	6	6	8	21
reward scale	100	1	3	1	3	3
observation embedding dimension	4	6	12	12	16	42
scale/translation hidden units	2	3	6	6	8	21

policies use Gaussian action prior. For the Ant maze task, the latent sample of the high-level policy is sampled once in the beginning of the rollout and kept fixed until the next one. The same high-level action is kept fixed over three environment steps. Otherwise, all the policy parameters for the high-level policies are equal to the benchmark parameters.

The environments used for training the low-level policies are otherwise equal to the benchmark environments, except for their reward function, which is modified to yield velocity based reward in any direction on the xy-plane, in contrast to just positive x-direction in the benchmark tasks. In the Ant maze environment, the agent receives a reward of 1000 upon reaching the goal and 0 otherwise. In particular, no velocity reward nor any control costs are awarded to the agent. The environment terminates after the agent reaches the goal.

Appendix D

Additional Results

D.1 Soft Q-Learning

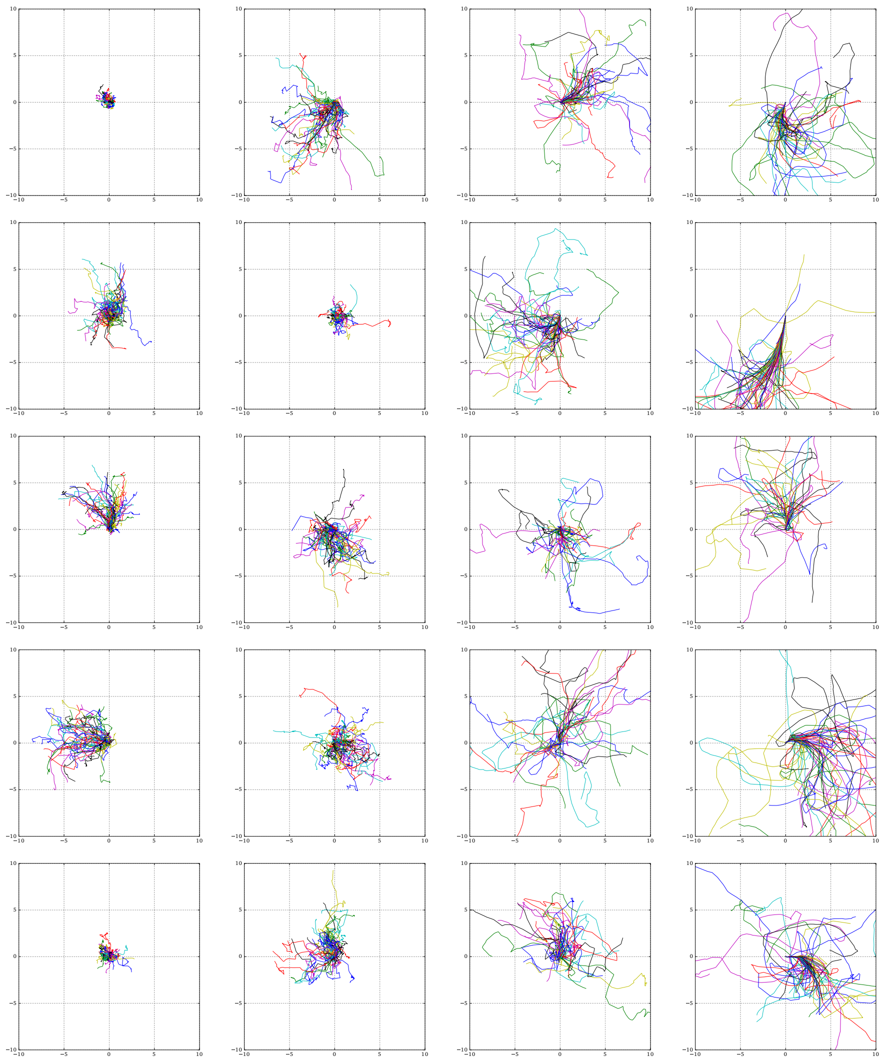


Figure D.1: The plot shows trajectories of the quadrupedal robot during maximum entropy pretraining. The robot has diverse behavior and explores multiple directions. The four columns correspond to entropy coefficients $\alpha = 10, 1, 0.1, 0.01$ respectively. Different rows correspond to policies trained with different random seeds. The x and y axes show the x and y coordinates of the center-of-mass. As α decreases, the training process focuses more on high rewards, therefore exploring the training ground more extensively. However, low α also tends to produce less diverse behavior. Therefore the trajectories are more concentrated in the fourth column.

D.2 Soft Actor-Critic

Figure D.2 compares SAC to Trust-PCL (Nachum et al., 2018), which learns slower than SAC but can eventually solve the tasks if ran longer. The figure also includes two variants of SAC: a variant that periodically copies the target value network weights directly instead of using exponentially moving average, and a deterministic ablation which assumes a deterministic policy in the value update and the policy update (Equation 4.9), and thus strongly resembles DDPG with the exception of having two Q-functions, using hard target updates, not having a separate target actor, and using fixed exploration noise rather than learned. Both of these methods can learn all of the tasks and they perform comparably to SAC on all but Humanoid (rllab) task, on which SAC is the fastest.

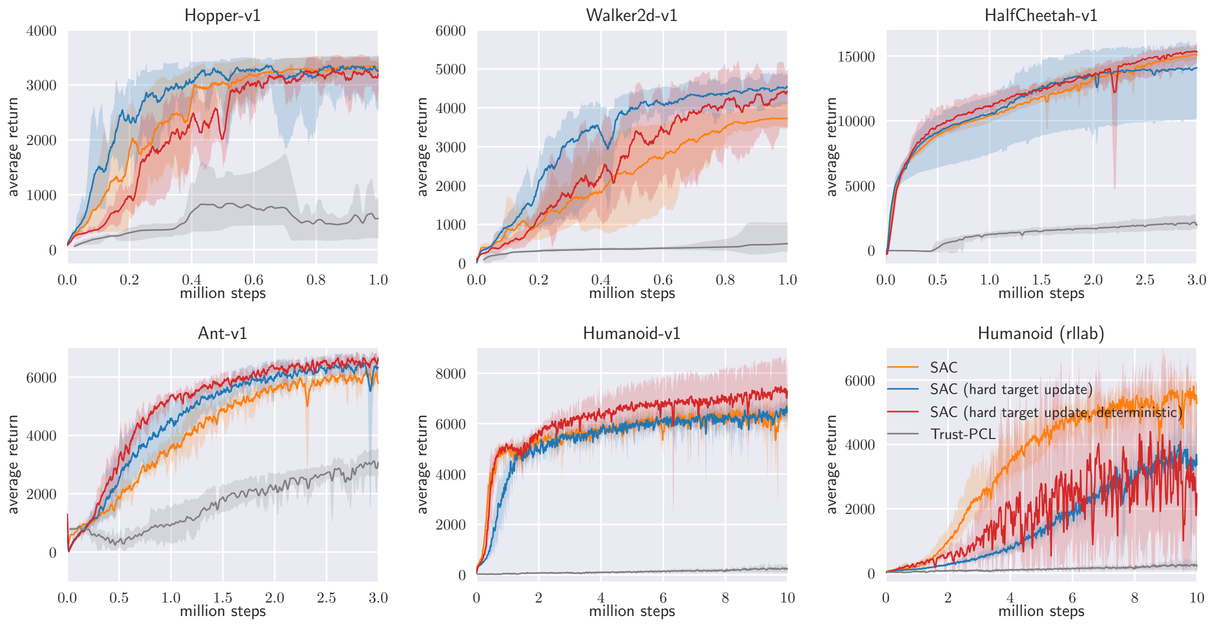


Figure D.2: Training curves for additional baseline (Trust-PCL) and for two SAC variants. Soft actor-critic with hard target update (blue) differs from standard SAC in that it copies the value function network weights directly every 1000 iterations, instead of using exponentially smoothed average of the weights. The deterministic ablation (red) uses a deterministic policy with fixed Gaussian exploration noise, does not use a value function, drops the entropy terms in the actor and critic function updates, and uses hard target updates for the target Q-functions. It is equivalent to DDPG that uses two Q-functions, hard target updates, and removes the target actor.