Actor-Expert: A Framework for using Q-learning in Continuous Action Spaces

by

Sungsu Lim

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Q-learning can be difficult to use in continuous action spaces, because a difficult optimization has to be solved to find the maximal action. Some common strategies have been to discretize the action space, solve the maximization with a powerful optimizer at each step, restrict the functional form of the action-values, or optimize a different entropy-regularized objective to learn a policy proportional to action-values. Such methods however, can prevent learning accurate action-values, be expensive to execute at each step, or find a potentially suboptimal policy. In this thesis, we propose a new policy search objective that facilitates using Q-learning and a new framework called Actor-Expert, that optimizes this objective. The Expert uses approximate Q-learning to update the action-values towards optimal action-values. The Actor iteratively learns the maximal actions over time for these changing action-values. We develop a Conditional Cross Entropy Method (CCEM) for the Actor, where such a global optimization approach facilitates use of generically parameterized action-values (Expert) with a separate policy (Actor). This method iteratively concentrates density around maximal actions, conditioned on state. We demonstrate in a toy environment that Actor-Expert with unrestricted action-value parameterization and efficient exploration mechanism succeeds while previous Q-learning methods fail. We also demonstrate that Actor-Expert performs as well as or better than previous Q-learning methods on benchmark continuous-action environments. We also show that it is comparable against Actor-Critic baselines, suggesting a new distinction among methods that learn both value function

and policy: learning action-values of the current policy or (optimal) action-values decoupled from the policy.

Preface

Parts of this thesis have been submitted as a journal paper. Lei Le and Yangchen Pan helped implement two baseline algorithms (PICNN and NAF) for the experiments in chapter 5. The rest is original work done by Sungsu Lim.

Success is often achieved by those who don't know that failure is inevitable.
(like our RL agents)

– Coco Chanel

Acknowledgements

I want to first and foremost thank my supervisor Martha White, for guiding me throughout my studies. She has been very patient and understanding in guiding me through research with continuous words of encouragement. I would not have been able to make it this far without her as my mentor. Whenever I was stuck on a problem, instead of providing answers she would ask back questions allowing me to organize my thoughts and provoking a logical thought process that would lead me to a solution.

I would like to thank the members of the RLAI lab, especially the #boba_guys in their friendship and support during my journey. Our frequent bubble tea breaks, the discussions we had, and the late nights spent together in the lab, are all very fond memories to look back upon, which has made this journey much more enjoyable. I also want to thank my friend Muhammad Zaheer whom I have learned a lot from since we first began our studies together.

Finally, I would like to thank my parents back in Korea who have provided me with endless support and helped me grow in every aspect possible. I am very fortunate to have them as my parents, giving me the privilege to explore various parts of the world at an early age as well as the freedom to seek my passion.

I am indebted to all these people who have helped me so far and I hope I too can pass on all the help I have received.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The physical world we live in is inherently continuous. In order to operate well in this world, it is imperative that the intelligent tools/agents we develop are able to handle continuous actions effectively, such as in robotics and industrial control applications. Not only is it important to learn in continuous action spaces, it can also be advantageous due to smooth generalization across actions. In this continuous action setting however, the action-selection process can be expensive. Action-selection needs to be efficient to ensure that agents are reactive and can make decisions quickly. This criteria is a deciding factor when selecting between *value-based algorithms* — those that implicitly specify a policy based on learned value functions — and *policy-based algorithms* — those that directly learn a policy.

Policy-based methods are more commonly used for continuous actions, because action-selection is fast for commonly chosen policy parameterizations. Policy-based methods explicitly optimize policy parameters, according to an objective based on cumulative reward under the policy. They include policy gradient methods like REINFORCE (Williams 1992), Actor-Critic (Degris, Pilarski, *et al.* 2012; Sutton 1984) and also non-gradient based methods, such as Cross Entropy for Policy Search (Mannor *et al.* 2003), CMA-ES (Hansen *et al.* 2003), and Reward Weighted Regression (Peters *et al.* 2007). A common policy parameterization used with these policy-based methods is a Gaussian distribution over actions, conditioned on a (complex) function of state. Action-selection simply corresponds to sampling from this distribution, which

is efficient.

Q-learning, on the other hand, only learn action-values which implicitly specify a policy. Action-selection at each step corresponds to solving a potentially difficult optimization problem over the continuous action space according to the action-values. Q-learning though, has several advantages over current policy-based methods. Q-learning is more natural to use off-policy, because the agent attempts to find the best action by estimating action-values for the optimal policy rather than the current policy. Off-policy updating enables the use of past data with experience replay, which can significantly improve sample efficiency. Many policy-based methods, on the other hand, are on-policy (Bhatnagar *et al.* 2008; Degris, Pilarski, *et al.* 2012; Mnih *et al.* 2016; Sutton 1984), or use an approximate[1] off-policy update (Degris, White, *et al.* 2012; Lillicrap *et al.* 2016; Silver *et al.* 2014; Wang *et al.* 2017). Secondly, empirical evidence suggests value-based methods can be more effective in certain problems. In the game of Atari, for example, a policy gradient method with many optimizations called ACER (Wang *et al.* 2017), needs parallel environment simulations to match the performance of Q-learning with experience replay. Deep Deterministic Policy Gradient (DDPG) (Lillicrap *et al.* 2016) achieves high performance in some benchmark environments, but also seems to suffer from hyperparameter sensitivity (Duan *et al.* 2016; Henderson *et al.* 2017).

In this thesis, we explore how we can use sample-efficient value-based methods like Q-learning in continuous action spaces, by making action-selection more efficient like policy-based methods.

1.1 Contribution

The list of contributions in this thesis is as follows:

- We develop a new algorithmic Q-learning framework with a policy, called Actor-Expert, that optimizes a new policy search objective. Action-selection becomes fast with an explicit Actor, and the learned action-values are unre-

[1] Sound off-policy variants have only been derived recently (Imani *et al.* 2018; Maei 2018), with as yet little empirical investigation into their efficacy.

stricted, more decoupled from the policy compared to Actor-Critic methods. (Section 4.1, 4.2)

- We introduce an instance of the Actor-Expert framework using a Conditional Cross Entropy Method (CCEM) for the Actor, that slowly learns maximal actions over time. We extend the global optimization algorithm, the Cross Entropy Method (Rubinstein 1999), to be conditioned on states so that Actor iteratively increases the likelihood of near-maximal actions for the Expert over time. (Section 4.3)

- We design a toy domain to highlight issues with existing Q-learning methods designed for these continuous actions spaces. (Section 6.1)

- We provide an empirical comparison of Q-learning methods in continuous action spaces, in several benchmark domains. (Section 8.1)

- We provide a preliminary comparison of methods that learn both a value-function and policy, comparing Actor-Expert to Actor-Critic algorithms. (Section 6.2, 8.2)

Chapter 2

Background

This chapter explains fundamental concepts and notations in Reinforcement Learning used frequently throughout the thesis. The first section covers the basic notion and objective of Reinforcement Learning. The subsequent two sections describe the two main branches of model-free Reinforcement Learning for Control. Readers already familiar with Reinforcement Learning may skip this chapter.

2.1 Reinforcement Learning Objective

In the Reinforcement Learning problem setting, an agent interacts with the environment and receives some reward. The goal of the agent is to collect as much reward as possible, maximizing the cumulative sum of rewards by learning the optimal sequence of actions.

More formally, this interaction between the agent and the environment can be formalized as a Markov decision process (MDP) specified by the tuple $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$. $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space, with $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty]$ defining the one-step state transition dynamics, and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ defining the reward function. $\gamma \in [0, 1)$ is the discount factor, indicating how much importance we put in delayed future rewards.

At each time step t, the agent is in a state $S_t \in \mathcal{S}$ and takes an action $A_t \in \mathcal{A}$. After taking the action, the environment transitions the agent to the next state S_{t+1} and gives a scalar reward R_{t+1} according to P and R respectively. The interaction between the agent and the environment is depicted in Figure

2.1.

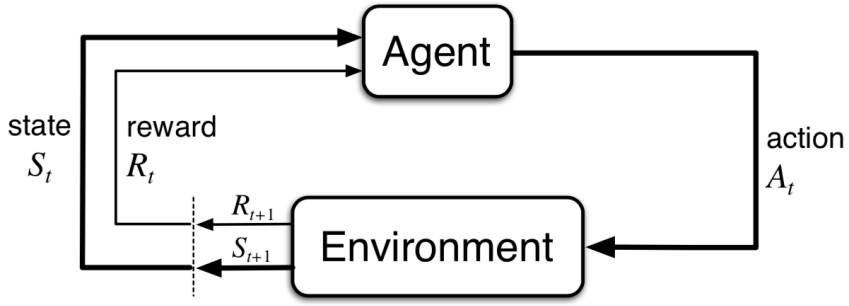

Figure 2.1: The agent-environment interaction in a Markov decision process. Figure adapted from Sutton and Barto (2018).

The discounted return (sum of discounted rewards) given at each time step is defined as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

The discount factor $\gamma \in [0, 1)$ is used to indicate preference for immediate rewards over future rewards.

The objective of the agent can now be formally defined as maximizing the expected discounted return G_t.

$$\max_{\pi} \mathbb{E}[G_t] \tag{2.2}$$

In order to maximize this, the agent has to learn a good policy π – a mapping from states to a probability distribution of selecting one of the possible actions – that would lead to states that generate good rewards, not only immediately but also in the long term.

There are two branches of learning this policy: value-based methods and policy-based methods. Subsequent sections will explain and highlight the differences between these two branches.

2.2 Value-based Methods

In value-based methods, agents learn value functions that encode a notion of good states, states that lead to high return. The state-value function of state

s under policy π, is defined as:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \text{ for all } s \in \mathcal{S} \tag{2.3}$$

Similarly, the action-value function of state-action tuple (s, a) under policy π is defined as:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{2.4}$$

The action-value function $q_\pi(s, a)$ explicitly includes the effect of taking an action, and is more commonly used for control. These value functions all satisfy a recursive relationship, also known as the Bellman equation (2.5), that is used to iteratively learn and improve the value estimates. From Eq. 2.4, the Bellman equation for q_π is:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \int_{\mathcal{S}} P(s'|s, a)\left[R(s, a, s') + \gamma q_\pi(s', a')\right] ds' \end{aligned} \tag{2.5}$$

For control however, the goal is not to learn the accurate value function for a certain policy π but to learn the optimal value function of the optimal policy π_*. The action-value function of the optimal policy, denoted q_*, is defined as:

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \tag{2.6}$$

This optimal action-value function also satisfies a recursive relationship, and it is referred to as the Bellman optimality equation:

$$q_*(s, a) \doteq \int_{\mathcal{S}} P(s'|s, a)\left[R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} q_*(s', a')\right] ds. \tag{2.7}$$

Using Eq. 2.7, the optimal action-value function is typically learned iteratively through Q-learning (Watkins 1989).

Action-values are learned as a parameterized function, where given a state-action pair the function outputs a corresponding action-value. Action-value functions can be parameterized by many different function approximators from

simple linear features to complex neural networks. Given an action-value function Q_θ parameterized by $\theta \in \mathbb{R}^n$, the general Q-learning update with function approximation is:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha \delta_t \nabla_\theta Q_\theta(S_t, A_t) \\ \text{where } \delta_t &\doteq R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_\theta(S_{t+1}, a') - Q_\theta(S_t, A_t) \end{aligned} \tag{2.8}$$

The Q-learning update with function approximation (Eq. 2.8) can be interpreted as updating θ in a direction that minimizes a projected Bellman error δ.

Q-learning is an off-policy algorithm, which learns the action-values for the optimal policy while following a different behavior policy. After the optimal action-value function is learned, the optimal policy is implicitly defined by taking actions greedily — selecting actions with the highest action-value at each state.

Q-learning can be easily generalized to continuous state spaces but not to continuous action spaces. Due to the max operator in both the Q-learning update and the implicitly induced policy, two difficult optimizations need to be solved at each step. $Q_\theta(s, \cdot)$ cannot be queried for all actions, and the optimization can be difficult to solve, especially if $Q_\theta(s, \cdot)$ is non-concave in a. For real-time agents in particular, decisions need to made quickly at least during the execution of policy, and the delay from the optimization over continuous actions is impractical.

2.3 Policy-based Methods

The other branch called policy-based methods, tries to learn the policy directly instead of learning a value function. A policy $\pi_{\mathbf{w}}$ is parameterized by $\mathbf{w} \in \mathbb{R}^m$, and we typically use $\pi_{\mathbf{w}}$ to represent a simple Gaussian or Gaussian mixture distribution. In contrast to value-based methods, action selection simply corresponds to sampling from the distribution $\pi_{\mathbf{w}}(\cdot|S_t)$.

The policy is learned by optimizing the parameters $\mathbf{w}$ with respect to a certain performance measure $J(\mathbf{w})$. This objective can be optimized with

derivative-free methods (evolutionary methods) and policy gradient methods. In the scope of this thesis, we focus on policy gradient methods, as they are more efficient and scale well to the dimensionality of the policy parameters. We will look at off-policy policy gradient methods in particular, that optimize the off-policy objective (Degris, White, *et al.* 2012).

The off-policy objective $J(\mathbf{w})$ is defined as:

$$\begin{aligned} J(\mathbf{w}) &\doteq \int_{\mathcal{S}} d(s) \int_{\mathcal{A}} \mathbb{E}_{\pi_{\mathbf{w}}}[R_{t+1}|S_t = s, A_t = a]\pi_{\mathbf{w}}(a|s)\ da\ ds \\ &= \int_{\mathcal{S}} d(s) \int_{\mathcal{A}} q_{\pi_{\mathbf{w}}}(s,a)\pi_{\mathbf{w}}(a|s)\ da\ ds \end{aligned} \tag{2.9}$$

where $d(s)$ is a different distribution from $\pi_{\mathbf{w}}$, usually set as the start state distribution in episodic tasks. We wish to update $\mathbf{w}$ in a direction such that this objective $J(\mathbf{w})$ is maximized.

The gradient of this objective can be approximated as follows.

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{w}) &= \nabla_{\mathbf{w}} \int_{\mathcal{S}} d(s) \int_{\mathcal{A}} q_{\pi_{\mathbf{w}}}(s,a)\pi_{\mathbf{w}}(a|s)\ da\ ds \\ &= \mathbb{E}_{S_t \sim d}\left[\int_{\mathcal{A}} q_{\pi_{\mathbf{w}}}(S_t,a)\nabla_{\mathbf{w}}\pi_{\mathbf{w}}(a|S_t) + \nabla_{\mathbf{w}} q_{\pi_{\mathbf{w}}}(S_t,a)\pi_{\mathbf{w}}(a|S_t)\ da\right] \end{aligned} \tag{2.10}$$

$$\begin{aligned} &\approx \mathbb{E}_{S_t \sim d}\left[\int_{\mathcal{A}} q_{\pi_{\mathbf{w}}}(S_t,a)\nabla_{\mathbf{w}}\pi_{\mathbf{w}}(a|S_t)\ da\right] \\ &= \mathbb{E}_{S_t \sim d}\left[\int_{\mathcal{A}} \pi_{\mathbf{w}}(a|S_t) q_{\pi_{\mathbf{w}}}(S_t,a)\frac{\nabla_{\mathbf{w}}\pi_{\mathbf{w}}(a|S_t)}{\pi_{\mathbf{w}}(a|S_t)}\ da\right] \\ &= \mathbb{E}_{S_t \sim d, A_t \sim \pi_{\mathbf{w}}}\left[q_{\pi_{\mathbf{w}}}(S_t,A_t)\frac{\nabla_{\mathbf{w}}\pi_{\mathbf{w}}(A_t|S_t)}{\pi_{\mathbf{w}}(A_t|S_t)}\right] \\ &= \mathbb{E}_{S_t \sim d, A_t \sim \pi_{\mathbf{w}}}[q_{\pi_{\mathbf{w}}}(S_t,A_t)\nabla_{\mathbf{w}} \ln \pi_{\mathbf{w}}(A_t|S_t)] \\ &= \mathbb{E}_{S_t \sim d, A_t \sim \pi_{\mathbf{w}}}[G_t \nabla_{\mathbf{w}} \ln \pi_{\mathbf{w}}(A_t|S_t)] \end{aligned} \tag{2.11}$$

Degris, White, *et al.* 2012 omit the second term in Eq.2.10, which then becomes analogous to the on-policy policy gradient (Sutton, McAllester, *et al.* 2000). Also note that $q_{\pi_{\mathbf{w}}}$ is the true action-value function. An unbiased estimate of the gradient can be obtained using the sampled return G_t (Eq. 2.11), which forms the basis of REINFORCE method for on-policy case (Williams 1992). REINFORCE uses the Monte Carlo return G_t to update policy parameters, which can have high variance and learn slowly.

Instead, a more popular approach is to use bootstrapped estimates of the return to lower variance and accelerate learning. This is done by replacing G_t in Eq. 2.11 with one-step return $R_{t+1} + \gamma V_\theta^{\pi_\mathbf{w}}(S_{t+1})$. It is also common to subtract a state-based baseline, usually $V_\theta^{\pi_\mathbf{w}}(S_t)$. This method is called the Actor-Critic algorithm (Sutton 1984; Degris, Pilarski, *et al.* 2012), which learns an additional value function (Critic), where the Critic evaluates how good an action is under the current policy (Actor).

There are many variants of Actor-Critic using different bootstrapped estimates, and for our implementation we use Advantage function $Q_\theta^{\pi_\mathbf{w}}(S_t, A_t) - V_\theta^{\pi_\mathbf{w}}(S_t)$ in the Actor Update (Schulman, Moritz, *et al.* 2016). The update rule for this variant is shown below:

Given a policy $\pi_\mathbf{w}$ and action-value function $Q_\theta^{\pi_\mathbf{w}}$ parameterized by $\mathbf{w}, \theta \in \mathbb{R}^n$ respectively,

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_t^\mathbf{w}[Q_\theta^{\pi_\mathbf{w}}(S_t, A_t) - V_\theta^{\pi_\mathbf{w}}(S_t)]\nabla \ln \pi_\mathbf{w}(A_t|S_t) \\ \theta_{t+1} &= \theta_t + \alpha^\theta \delta_t \nabla_\theta Q_\theta(S_t, A_t) \\ &\text{where } \delta_t \doteq R_{t+1} + \gamma Q_\theta(S_{t+1}, A_{t+1}) - Q_\theta(S_t, A_t) \end{aligned} \tag{2.12}$$

These policy-based methods still learn value functions. The main distinction is that they learn an explicit policy. Further, the distinction from Q-learning is that the Critic learns the value function of the current policy rather than the optimal policy.

2.4 Conclusion

In this chapter we covered the basic concept of Reinforcement Learning and its objective. We then described two control strategies: value-based methods and policy-based methods. We explained why it can be difficult to apply Q-learning in continuous action spaces. In the following chapter, we will discuss previous attempts to apply Q-learning in continuous action spaces.

Chapter 3

Existing Q-learning methods in Continuous Action Spaces

In this chapter, we describe some existing approaches using Q-learning in continuous action spaces. Previous strategies can be categorized into three main types:

- Solving the maximization at each step
- Restricting the form of the action-values
- Using a soft Bellman operator

In the following sections we describe previous works related to each strategy.

3.1 Solving the maximization at each step

The first strategy solves the maximization at each step by using cleverly discretized action spaces or with a sufficiently powerful optimizer. Santamaría *et al.* (1998), Millán *et al.* (2002), Kimura (2007), and Metz *et al.* (2017) discretize the action space so that the maximum action can be found more easily. However, the performance would vary on the granularity of discretization and would not scale very well to higher dimensions.

Another approach is to use a sufficiently powerful optimizer that can effectively approximate the maximal action. The most general purpose approach is **QT-Opt** (Kalashnikov *et al.* 2018), which also uses the Cross Entropy Method

(CEM) (Rubinstein 1999) that iteratively finds the optimum at each step. QT-Opt performs a fixed number of iterations of CEM to find the approximate maximum action.

These methods can be straightforward to implement and effective in small dimensions. But maximizing the action at each step can become prohibitively expensive which is especially infeasible for real-world applications requiring efficient action-selection in high-dimensional spaces.

3.2 Restricting the action-value function

The second strategy restricts the class of action-value function so that the maximization is more tractable. This method facilitates efficient action selection without the need of a separate policy.

For simple problems, the policy can be parameterized as a linear function. Millington (1991) learns a linear policy, with the coefficients generated randomly from the probability density function parameterized by a radial basis function network. Differing from the discrete binnings in the previous approach, the radial basis function network allows a soft boundary over the continuous state space. However, this method would not scale well to complex problems requiring more radial basis and non-linear policy.

Wire-fitting (Baird and Klopf 1993; Gaskett *et al.* 1999) learns a set of action control points and its corresponding action-values $\mathcal{C} = \{(a_i, q_i) : i = 1, ..., m\}$ to approximate the action-value for each state. Action-values within the continuous action space are then approximated by interpolating between these set of points. This approximation forces the maximizing action to be one of these action points, which can be found easily. However, the quality of the action-value function is highly dependent on the number of control points, and does not scale well to high dimensions.

Normalized Advantage Function (NAF) (Gu, Lillicrap, Sutskever, *et al.* 2016) learns an action-value function that is constrained to be concave and quadratic in terms of actions. More concretely, the action-value function is decomposed into state-value function and advantage function (Baird 1993;

Harmon *et al.* 1996a; Harmon *et al.* 1996b), $q(S_t, A_t) \doteq v(S_t) + adv(S_t, A_t)$, where the advantage function $adv(S_t, A_t)$ is quadratic, and so concave. Due to this parameterization, maximum action can be found immediately.

Partial Input Convex Neural Networks (PICNN) (Amos *et al.* 2017) learns a more general function form than NAF. PICNN learns a concave action-value function with respect to actions by restricting weights and activations to be non-negative. The maximum action is found by an approximate gradient ascent procedure from a random starting point. The action-value function is more expressive than a quadratic, but the maximum action-selection process is more expensive.

The methods in this second strategy make maximization more feasible at the cost of restricting the learned action-value function. Wire-fitting assumes interpolated action-values between the control points to be linear (or that the control points are sufficiently granular) while NAF and PICNN assume the action-value to be unimodal, which can be quite restrictive.

3.3 Using a Soft Bellman Operator

The last strategy avoids using the maximum operator in the update by optimizing an entropy regularized objective. If the standard reinforcement learning objective was to find a policy that maximizes $\sum_t \mathbb{E}_{(s_t,a_t)\sim\pi}[R(s_t, a_t, s_{t+1})]$, the new entropy regularized objective is to find a policy that maximizes

$$\sum_t \mathbb{E}_{(s_t,a_t)\sim\pi}[R(s_t, a_t, s_{t+1}) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{3.1}$$

$\mathcal{H}(\pi(\cdot|S_t))$ is the entropy of the policy, and α is a temperature hyperparameter weighting the relative importance with respect to reward.

Methods using this entropy regularized objective learn soft action-values, and define a policy using energy-based models of these soft action-values. The policy is implicitly defined by these action-values, so it is still considered as a value-based method. More concretely, the policy takes the following form:

$$\pi(a_t|s_t) \propto exp(-\frac{1}{\alpha}q_\pi^{soft}(s_t, a_t)) \tag{3.2}$$

Then under this new objective, the optimal policy can be shown to be equivalent to the following (Haarnoja, Tang, *et al.* 2017):

$$\pi_*^{soft}(a_t|s_t) = exp(\frac{1}{\alpha}(q_*^{soft}(s_t, a_t) - v_*^{soft}(s_t))), \tag{3.3}$$

$$\text{where } q_*^{soft}(s,a) \doteq \int_{\mathcal{S}} P(s'|s,a)\left[R(s,a,s') + \gamma v_*^{soft}(s')\right]ds \tag{3.4}$$

$$v_*^{soft}(s') \doteq \alpha \log \int_{\mathcal{A}} exp(\frac{1}{\alpha} q_*^{soft}(s', a'))da' \tag{3.5}$$

Using the recursive soft optimal Bellman equation in Eq. 3.4, the $q_*^{soft}(s,a)$ can be learned without the intractable max operator. Integration over actions in Eq. 3.5 can be substituted by sampling:

$$\begin{aligned} v_*^{soft}(S_t) &\doteq \alpha \log \int_{\mathcal{A}} exp(\frac{1}{\alpha} q_*(S_t, a'))da' \\ &= \alpha \log \int_{\mathcal{A}} \pi(a'|S_t) \left[\frac{exp(\frac{1}{\alpha} q_*(S_t, a'))}{\pi(a'|S_t)}\right] da \\ &= \alpha \log \mathbb{E}_{a' \sim \pi(\cdot|S_t)} \left[\frac{exp(\frac{1}{\alpha} q_*(S_t, a'))}{\pi(a'|S_t)}\right] \end{aligned} \tag{3.6}$$

Soft Q-learning (SQL) (Haarnoja, Tang, *et al.* 2017) takes this approach. The policy is parameterized as the exponential function of the negative action-values (Gibbs distribution). Action-selection and the action-value update then involves sampling from this policy, and SQL learns a state-conditioned stochastic sampling network trained to approximate sampling from the energy-based distribution. Sampling from the energy-based distribution can be expensive, and sampling actions proportional to their action-values does not generate a robust maximizing action. Furthermore, using the soft Bellman equation optimizes over a different entropy-regularized objective; in cases where the traditional value-based objective is preferred, this may not be desirable.

3.4 Conclusion

In this chapter we looked at several approaches to use Q-learning in continuous action spaces. These methods however, put restrictions on action-value functions, or is still inefficient during action-selection, or optimize a regularized

objective. We wish to develop a Q-learning method with minimal assumptions on action-values and which enables efficient action-selection.

Chapter 4

The Actor-Expert Framework

In this chapter we introduce our main contribution, Actor-Expert, a new value-based Q-learning framework in continuous action spaces. The action-value function (Expert) is learned via (approximate) Q-learning. The policy (Actor) is learned by optimizing a new policy objective, different from the typical policy gradient objective in policy-based methods. By having a separate maximizing policy (Actor), the agent can select actions efficiently, foregoing the expensive maximization process and the need to restrict the action-value function.

In the following sections we begin by proposing a new policy search objective that this framework optimizes, and highlight its difference from the usual policy gradient objective. We then describe the Actor-Expert framework in detail, and introduce an instance of Actor-Expert by extending the Cross Entropy Method (CEM) (Rubinstein 1999) to be conditioned on states. We then draw distinctions between existing methods that explicitly model both action-values and a policy, such as Actor-Critic.

4.1 Actor-Expert Objective

In this section, we propose a new objective for the policy and contrast it to the typical policy gradient objective of Actor-Critic. The new objective relies on an expert to provide optimal action-value estimates. Given such estimates, the goal of the policy is simple: concentrate probability around high-valued actions in each state.

First consider learning a policy that returns the maximal actions for a

given true optimal action-values, q_*. For a deterministic policy and assuming a unique maximal action, the goal is to find π such that $\pi(s) = \arg\max_a q_*(s, a)$. In general, however, multiple actions could be optimal. Further, depending on the policy parameterization, it is unlikely that π can return the maximal action for every state. Rather, π will have to trade off accuracy across states.

More generally then, we can consider stochastic policies that approximately concentrate probability around maximal actions. We define the following weighted objective

$$J_{\text{AE}}(\mathbf{w}) \doteq \int_{\mathcal{S}} d(s) \int_{\mathcal{A}} q_*(s, a)\pi_{\mathbf{w}}(a|s)\ da\ ds \tag{4.1}$$

for some weighting $d : \mathcal{S} \to [0, \infty)$ over states. This weighting reflects the sampling distribution of states for updates, and impacts the trade-off in approximation. Typically, for off-policy objectives where data is gathered according to another behaviour policy, d is the stationary distribution for that behaviour policy.

The goal of the Actor $\pi_{\mathbf{w}}$ is to concentrate density on maximal actions within these states. For a policy that can perfectly represent a maximal action in each state, if $d(s) > 0$ for all $s \in \mathcal{S}$, then $\pi_{\mathbf{w}}$ will become highly peaked around the maximal action for each state. If the policy parameterization enables delta distributions, then the set of optimal policies will be those that put all density on the sets $\max_a q_*(s, a)$.

This objective appears similar to the corresponding off-policy policy gradient objective introduced earlier, using d from a different distribution:

$$J_{\text{AC}}(\mathbf{w}) \doteq \int_{\mathcal{S}} d(s) \int_{\mathcal{A}} q_{\pi_{\mathbf{w}}}(s, a)\pi_{\mathbf{w}}(a|s)\ da\ ds \tag{4.2}$$

In practice, for both objectives J_{AE} and J_{AC}, the true action-value function q_* and $q_{\pi_{\mathbf{w}}}$ are not available and we approximate these action-value functions by learning Q^*_θ and $Q^{\pi_{\mathbf{w}}}_\theta$ respectively. In Actor-Critic, the policy parameters are adjusted to increase the density of actions under the action-values for the current policy $Q^{\pi_{\mathbf{w}}}_\theta$. In Actor-Expert, the policy parameters are adjusted to increase the density of actions that are currently thought to be optimal under Q^*_θ. The naming reflects this difference: the Critic provides feedback about

the current policy whereas the Expert tells the policy which actions are high-valued.[1]

The distinction—using Q^*_θ rather than $Q^{\pi_\mathbf{w}}_\theta$—is subtle but results in fundamental differences. One difference is that the parameters for the action-values and the policy are decoupled in J_{AE}. In J_{AC}, changes to $\pi_\mathbf{w}$ both change the weighting on actions and the action-values $Q^{\pi_\mathbf{w}}_\theta$ themselves. For J_{AE}, however, changing $\pi_\mathbf{w}$ does not change Q^*_θ. We can take advantage of this decoupling to provide a global optimization strategy for the Actor, because for convergence we only need to show that the Actor tracks the changing Expert. Intuitively, another benefit is that the Expert only concentrates on identifying optimal action values, rather than wasting time evaluating a likely suboptimal policy. In fact, J_{AE} could incorporate any estimates identifying advantageous actions decoupled from the policy.

4.2 Actor-Expert Framework

Actor-Expert is a general framework for obtaining a policy that maximizes J_{AE} (Eq. 4.1). The framework is summarized in Algorithm 1. Key choices within this framework include: 1) how to approximate the Q-learning update; 2) how to obtain the maximum action from the Actor; 3) how to select actions for exploration; and 4) how to update the policy parameters according to J_{AE}. We provide specific choices for each of these components to make the algorithm more concrete.

The Expert uses an approximate Q-learning update for Q_θ. The exact Q-learning target is difficult to obtain in a continuous action setting without restricting action-values. Simply learning an explicit Actor does not solve the maximization problem inside Q-learning target, since the stochastic policy $\pi_\mathbf{w}$ only concentrates around maximal actions and is an approximation. Nonetheless, we can take advantage of the Actor to reduce the computation burden.

[1]The term Actor-Critic has at times been used whenever explicit policies and action-values are learned. For example, (Crites *et al.* 1995) show convergence of a modified Actor-Critic algorithm that uses Q-learning updates instead of Sarsa updates, with the goal to provide some theoretical characterization of Actor-Critic. In this work, we make the distinction highlighted above: a Critic does policy evaluation whereas an Expert does Q-learning.

Algorithm 1: Actor-Expert

Initialize Actor parameters $\mathbf{w}$ and Expert parameters θ.
for *t=1, 2, ...* **do**
  Observe S_t, take action A_t (e.g., $A_t \sim \pi_{\mathbf{w}}(\cdot|S_t)$), and observe R_{t+1}, S_{t+1}
  $a' \leftarrow$ estimated maximum action a' from Actor ▷ mode of $\pi_{\mathbf{w}}(\cdot|S_t)$
  $\bar{a}' \leftarrow$ gradient ascent on $Q_\theta(s, \cdot)$ starting from a' ▷ can use $\bar{a}' \leftarrow a'$
  Update Expert θ, using Q-learning with
  $\delta_t = R_{t+1} + \gamma Q_\theta(S_{t+1}, \bar{a}') - Q_\theta(S_t, A_t)$
  Update Actor $\mathbf{w}$ to maximize J_{AE} ▷ e.g., Algorithm 4

We could use the Actor to provide its current estimate of the maximal action, to use as a starting point to find the maximal action for the Q-learning update, such as by gradient ascent.

The simplicity of obtaining the maximal action from the Actor depends heavily on the policy parameterization. The Actor needs to return a mode of $\pi_{\mathbf{w}}(\cdot|s)$. For a unimodal Gaussian, with mean and covariance a (nonlinear) function of the state, the maximal action is simply the mean. For a bimodal Gaussian, there is not such a simple solution. One of the two means is likely to provide a reasonable approximation to a mode of the distribution. For a more careful solution, a gradient ascent could be started from each of the two means, which often results in global solutions (Carreira-Perpiñán 2000). In our own experiments, we found using the mean with the higher coefficient in the bimodal mixture model to be almost as effective; with more Gaussians in the mixture, a more careful solution might be necessary.

To take exploratory actions, since the Actor $\pi(\cdot|S_t)$ is a stochastic policy, we can simply sample from this policy. Note however that the framework is agnostic to the exploration mechanism, and any other exploration methods can be used.

The Actor can use a variety of updates to maximize J_{AE} for the given action-values Q^*_θ. In the next section, we provide an instance of Actor-Expert by extending a global optimization strategy Cross-Entropy Method conditioned on states that optimizes J_{AE}. We then conclude the chapter by drawing some connections and distinctions to related methods that explicitly learn both

value function and policy.

4.3 Conditional Cross Entropy Method for the Actor

In this section, we develop a global optimization algorithm for the Actor, based on the Cross Entropy Method (CEM) (Rubinstein 1999). Global optimization strategies are designed to find the global optimum of a general function $f(\theta)$ for some parameters θ. For example, for parameters θ of a neural network, the objective function f may be the loss function on a sample of data. The advantage of these methods is that they do not rely on gradient-based strategies, which are prone to getting stuck in local optima. Instead, they use randomized search strategies that have optimality guarantees in some settings (Hu *et al.* 2012) and have also been shown to be effective in practice (Hansen *et al.* 2003; Peters *et al.* 2007; Salimans *et al.* 2017; Szita *et al.* 2006).

4.3.1 Cross Entropy Method

Cross Entropy Method (CEM) (Rubinstein 1999) maintains a distribution $p(\theta)$ over parameters θ, iteratively narrowing the range of plausible solutions. The goal is to minimize the KL divergence between $p(\theta)$ to the uniform distribution over parameters where the objective function f is maximal: $I(f(\theta) \geq f^*)$ for $f^* = \max_\theta f(\theta)$. Practically, we do not have the threshold f^* nor the uniform distribution $I(f(\theta) \geq f^*)$. Rather, the algorithm approximates both iteratively.

At each iteration t, under the current threshold f_t, the thresholded uniform distribution is approximated with an empirical distribution, by sampling several parameter vectors $\theta_1, \ldots, \theta_N$, and keeping those $\theta_1^*, \ldots, \theta_h^*$ with $f(\theta_i^*) \geq f_t$ and discarding the rest. Normally f_t is set as the top $x\%$ quantile value. KL divergence is minimized between p_t and this empirical uniform distribution $\hat{I} = \{\theta_1^*, \ldots, \theta_h^*\}$, for $h < N$.

This step corresponds to maximizing the likelihood of θ in set $\hat{I}$ under distribution p_t. Iteratively, the distribution p_t over θ narrows around higher

valued θ. Sampling θ again from p_t narrows the search over θ and makes it more likely to produce a useful approximation to $I(f(\theta) \geq f_t)$.

An example of one iteration of CEM update is shown in Figure 4.1. From $p_t(\theta)$, 10 samples $(\theta_1, \ldots, \theta_{10})$ are selected. The thresholded uniform distribution is approximated by taking the top 20% quantile of the samples (θ_9 and θ_{10}). $p_t(\theta)$ is updated such that it is more likely to sample the top 20% quantile samples at the next iteration.

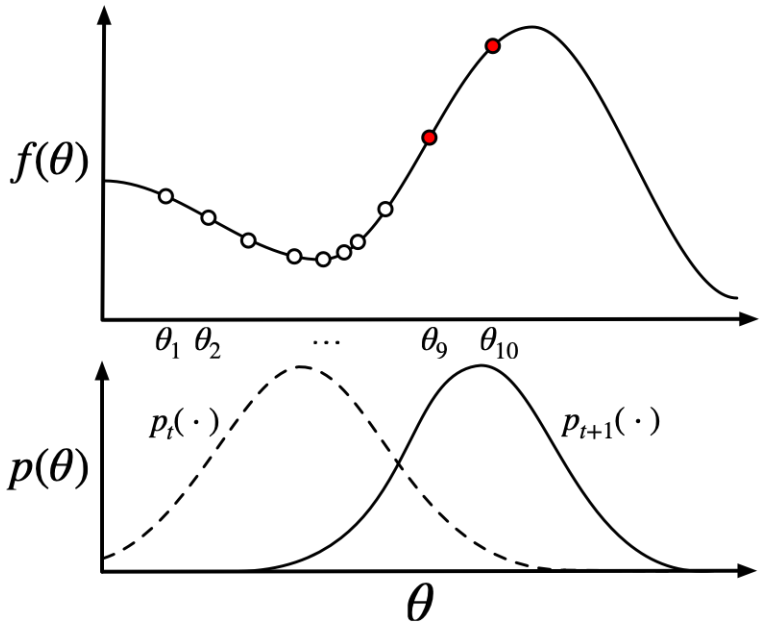

Figure 4.1: One iteration of CEM update

To make it more likely to find the global optimum, the distribution p_0 is initialized as a wide distribution, such as a Gaussian distribution with zero mean $\mu_0 = \mathbf{0}$ and a diagonal covariance $\mathbf{\Sigma}_0$ of large magnitude. The threshold f_t is implicitly increased over time, using upper quantiles as further explained below.

CEM, although effective at finding the optimal set of parameters, only finds the the optimal set parameters for a single optimization problem. Most of the works using CEM in reinforcement learning have been aiming to learn a single-best set of parameters that optimize towards higher roll-out returns (Mannor *et al.* 2003; Szita *et al.* 2006). Our goal, however, is to use CEM (repeatedly) to find maximizing actions a^* at each state for $Q(S_t, \cdot)$, rather than maximizing

parameters θ^* for $\mathbb{E}_{\pi_\theta}[G_0]$ which is a single global optimization over returns.

Technically CEM could be run on each step to find the exact maximal action for each current state, as QT-Opt (Kalashnikov *et al.* 2018) did, but this is expensive and throws away prior useful information about the function surface obtained when previous optimizations were executed.

4.3.2 Conditional Cross Entropy Method

We extend CEM to be (a) conditioned on state and (b) learned iteratively over time. CEM is well-suited to be extended to a conditional approach for use in the Actor, for two reasons. First and foremost, it is designed to be effective for general (non-concave) functions, and the action-values are likely non-concave with respect to actions. Second, it provides a natural stochastic Actor, since it maintains a distribution $p(\cdot)$ over high-valued actions for each state. Sampling from this distribution also provides a reasonable mechanism for exploration.

Conditional CEM (CCEM) iteratively executes CEM updates, across states. To do so, it replaces the learned $p(\cdot)$ with $\pi(\cdot|S_t)$, where $\pi(\cdot|S_t)$ can be any parametrized, multi-modal distribution. For a mixture model for example, the parameters would be conditional means $\mu_i(S_t)$, conditional diagonal covariances $\Sigma_i(S_t)$ and coefficients $c_i(S_t)$, for the ith component of the mixture.

At each step, we sample from the conditional mixture model, $\pi_{\mathbf{w}_t}(\cdot|S_t)$, to provide a set of actions $a_1, \ldots, a_N$ from which we construct the empirical distribution $\hat{I}(S_t) = \{a_1^*, \ldots, a_h^*\}$ with $Q_{\theta_t}(S_t, a_i^*) > f_t$. The policy parameters $\mathbf{w}_t$ are updated using a gradient ascent step on the log-likelihood of the actions in $\hat{I}(S_t)$.

The key step in this algorithm is to select the empirical distribution. A standard strategy for CEM is to use the top quantile, which both avoids the need to pick a threshold and provides a consistent number of points h for the likelihood step. For $a_1, \ldots, a_N$ sampled from $\pi_{\mathbf{w}_t}(\cdot|S_t)$, we select $a_i^* \subset \{a_1, \ldots, a_N\}$ where $Q(S_t, a_i^*)$ are in the top $(1-\rho)$ quantile values. For example, for $\rho = 0.2$, approximately top 20% of actions are chosen, with $h = \lceil \rho N \rceil$. Implicitly, f_t is $Q_\theta(S_t, a_h^*)$ where a_h^* is the action with the lowest value in this top quantile. This procedure is summarized in Algorithm 2.

There could be other strategies that produce a better estimate of the thresholded uniform distribution. Consider action-values that are differentiable in action. We can improve on the above procedure by performing a small number of gradient ascent steps from a_i to reach actions $\tilde{a}_i$ with slightly higher action-values. We could then select the top quantile of these improved actions. The resulting empirical distribution would contain actions with higher action-values on which to perform maximum likelihood. This additional gradient ascent can be expensive, but could also reduce the number of actions that need to be sampled. This procedure is summarized in Algorithm 3.

Algorithm 2: Quantile Empirical Distribution

Evaluate and sort in descending order:
$Q_\theta(S_t, a_{i_1}) \geq \ldots \geq Q_\theta(S_t, a_{i_N})$
▷ get top $(1-\rho)$ quantile, e.g. $\rho = 0.2$
return $\hat{I}(S_t) = \{a_{i_1}, \ldots, a_{i_h}\}$ (where $h = \lceil \rho N \rceil$)

Algorithm 3: Optimized Quantile Empirical Distribution

Input: n = number of gradient descent steps (e.g., $n = 10$)
for $i = 1, \ldots, N$ **do**
  $\tilde{a}_i \leftarrow$ perform n steps of gradient ascent starting from $Q_\theta(S_t, a_i)$
return Quantile Empirical Distribution($\{\tilde{a}_1, \ldots, \tilde{a}_N^*\}$)

The full update for the Actor using CCEM is summarized in Algorithm 4. This algorithm fits within the generical Actor-Expert framework described earlier in Algorithm 1. The CCEM update also makes the Actor focus its density around high-valued actions, weighted by action-values from the Expert. However, slightly different from the original proposed objective, CCEM weighs each actions above a threshold uniformly, potentially providing a more stable update from the approximation errors of the Expert.

In this subsection we have extended CEM to be conditioned on states. CEM is a sound approach to iteratively find the maximum for non-convex functions, but it can be unclear whether the Actor using CCEM would behave like the expected CEM optimizer across states. Readers are referred to Lim *et al.* (2018) where a detailed analysis and the formal proof is provided.

Algorithm 4: Conditional CEM for the Actor

Input: S_t and Q_θ
Sample N actions $a_i \sim \pi_{\mathbf{w}}(\cdot|S_t)$
Obtain empirical distribution $\hat{I}(S_t) = \{a_1^*, \ldots, a_h^*\}$
by evaluating $a_1, \ldots, a_N$ on Q_θ
▷ Increase likelihood for high-value actions
$\mathbf{w} \leftarrow \mathbf{w} + \alpha \frac{1}{N} \sum_{j=1}^{N} \mathbb{1}_{a_j \in \hat{I}(S_t)} \nabla_{\mathbf{w}} \ln \pi_{\mathbf{w}}(a_j|S_t)$

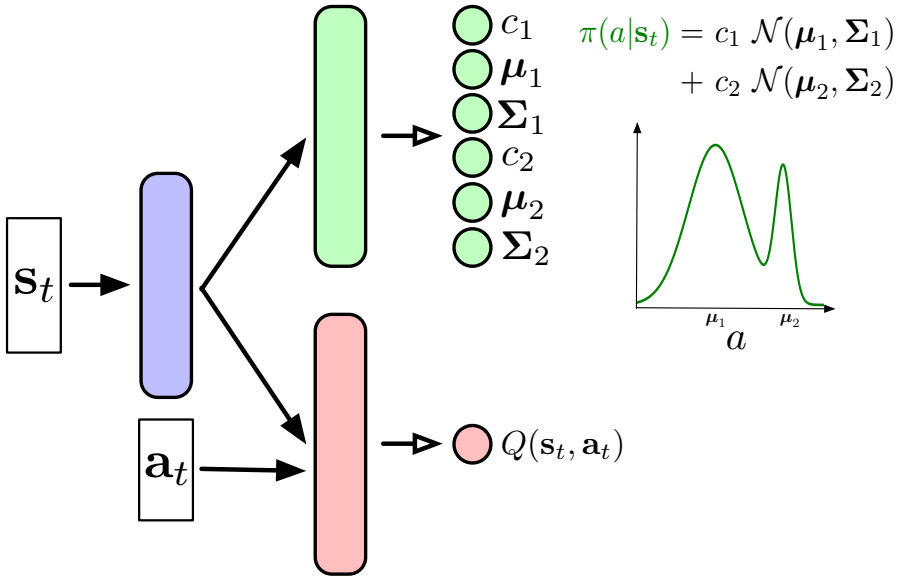

Figure 4.2: Actor-Expert with an Actor learning a bimodal Gaussian mixture. The Actor learns the conditional mixture distribution $\pi_{\mathbf{w}}(\cdot|S_t)$ parameterized with coefficients c_i, means μ_i an diagonal covariances Σ_i. Such multimodal stochastic Actor provides a natural exploration mechanism to gather data for the Expert. The Expert learns the action-value function $Q_\theta(S_t, A_t)$, where the action comes in through late fusion.

We conclude with a description of the actual implementation of Actor-Expert using CCEM. Figure 4.2 depicts an example architecture of Actor-Expert. The Actor and the Expert share the first neural network layer to learn a shared representation of the state, and learn separate functions conditioned on that state. Here we could have used separate networks for Actor and Expert, but chose to merge the two networks to reduce the number of parameters to closely match other value-based baseline methods.

The Actor parameterizes the policy using a Gaussian mixture model. To prevent the diagonal covariances $\boldsymbol{\Sigma}$ from exploding or vanishing, we bound

it between $[e^{-2}, e^2]$ using a tanh layer. To obtain the approximate maximal action from the Actor, we simply use the mean with the highest coefficient. The generic Actor-Expert algorithm shows online updates for each new sample, but experience replay can be easily incorporated storing observed data and updating with mini-batches from the buffer at each time step. The update to Q_θ is generic, and can incorporate modifications to Q-learning, like target networks. We do not, however, use target networks for updating $\pi_{\mathbf{w}}$.

4.4 Relationship to Other Algorithms with Explicit Policies and Action-values

In previous sections we have described the Actor-Expert objective and framework, along with an instance of Actor-Expert using Conditional Cross Entropy Method. Here we explain its distinction with other methods that also learn an explicit value function and a policy.

Soft Q-learning (SQL) mentioned in Chapter 3 explicitly learns a policy to sample actions proportional to the soft action-values (Haarnoja, Tang, *et al.* 2017), while the approximate optimal action-value function is learned using the soft optimal bellman operator avoiding the hard maximization problem. Follow-up work on Soft Actor-Critic (SAC) (Haarnoja, Zhou, *et al.* 2018) uses policy iteration, where the action-value function for the current policy is learned using soft Sarsa algorithm.[2]

Both SQL and SAC maintain an explicit policy that learns the distribution proportional to their learned soft action-values. They minimize the KL divergence between the parameterized policy and the energy distribution of their action-value function. The energy distribution would concentrate around high-valued actions, and consequently the Actor for SQL and SAC would also concentrate around high-valued actions similar to the Actor-Expert. However, the policies are inherently tied to the energy distribution with the level of concentration determined by the temperature parameter in their soft value

[2]The algorithm is unlike most Actor-Critic algorithms because it does not attempt to approximate the policy gradient. Rather, it is a Sarsa algorithm that learns an explicit parameterized policy to simplify the greedification step when learning with continuous actions.

updates. The proportionally sampled action from SQL can be a poor approximation of the maximizing action. Actor-Expert on the other hand allows for any policy that optimizes J_{AE} in Eq. 4.1.

SQL is closer to the Actor-Expert framework than SAC in that SQL attempts to learn the approximate soft optimal action-value while SAC learns action-value for the current policy. However both methods optimize for the entropy regularized objective; in some cases the optimal solution may be the same as the original objective but unlikely to hold in general.

There are several Actor-Critic methods that use (approximate) Q-learning updates, making the algorithms look similar to Actor-Expert algorithms, such as Deterministic Policy Gradient (DPG) (Silver *et al.* 2014) and NFQCA (Hafner *et al.* 2011). They are both policy gradient methods, but because the policy is deterministic, (a) the action-value update resembles a Q-learning update and (b) the policy update changes the policy to output a higher-value action according to its current action-values.

The Actor in DPG can be seen as an approximate maximizer, similar to the Actor in Actor-Expert, as it executes gradient ascent on the action-values. Because π is updated to output higher-valued actions, π has been regarded as an approximate maximizer, and thus the policy evaluation step using $A_{t+1} = \pi(S_{t+1})$, as approximate Q-learning. Nonetheless, because the update still uses the action generated by π directly, it is still better thought of as policy evaluation — Sarsa — update. A variant of DPG using $\pi(S_{t+1})$ such that the action from the policy and the maximal action for the action-value upate is more decoupled, would make this more explicitly an approximate Q-learning update.

Overall, most of these related methods are estimating Q^π for the action-values, whereas our proposal in Actor-Expert is to explicitly consider action-values that attempt to estimate the optimal action from a state, potentially agnostic to the current policy. When approximations are introduced, or deterministic policies are learned, there can be some overlap in the methods, despite different intents. The method that is most similar to the intent behind Actor-Expert is SQL, because it explicitly uses a (soft) Q-learning update and

learns a policy to take actions according to those action-values.

4.5 Conclusion

This chapter presented the Actor-Expert framework that facilitates the use of Q-learning in continuous action spaces. Different from previous Q-learning methods, the framework decouples action-value function and action-selection by introducing a separate policy (Actor). This Actor optimizes a new policy search objective different from conventional policy gradient objective, concentrating probability on high-valued actions based on the estimates given by the Expert (action-value function). The framework also decouples the action-values learned from the policy, so that a more optimal action-value function can be learned, potentially agnostic to the current policy. Using our proposed Actor-Expert with CCEM, we provide empirical results to show its advantages in the following chapter.

Chapter 5

Experiment Setup

In Chapters 6 to 9 we provide empirical results for previous Q-learning methods and Actor-Expert, as well as an investigation into choices in Actor-Expert. In this chapter we provide details on the experimental setup, describing the environments and algorithms used in the experiments. We design a toy environment to highlight issues with restricting the action-value function. We also evaluate all methods on four higher-dimensional benchmark environments, to validate that Actor-Expert performs comparably to other methods in more complex problems.

5.1 Environments

We first design a bimodal toy environment to highlight the limitation of restricting the functional form of action-values. This toy environment has a single state S_0 and $a \subset [-2, 2]$, where the true q^*—shown in Figure 5.1—is a function of two radial basis functions centered at $a = -1.0$ and $a = 1.0$, with an unequal reward of 1.0 and 1.5 respectively. This is a bandit setting with a single state, so the true optimal action-value function is equal to the expected reward.

For high-dimensional benchmark domains, we use one environment from OpenAI Gym (Brockman *et al.* 2016)—Pendulum—and three environments from MuJoCo (Todorov *et al.* 2012)—HalfCheetah, Hopper and Ant. The environment properties are summarized in Table 5.1. All domains are episodic, with discount factor $\gamma = 0.99$.

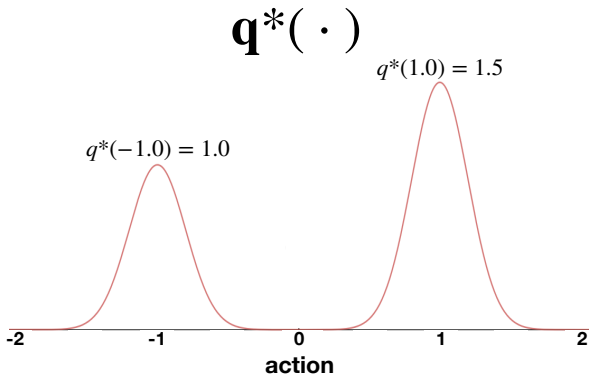

Figure 5.1: Optimal Action-values for the Bimodal Environment.

Environment	State dim.	Action dim.	Action range	Description
Pendulum (version v0)	3	1	[-2,2]	The goal of the agent is to swing the pendulum up to the top and stay upright, spending least amount of energy.
HalfCheetah (version v2)	17	6	[-1,1]	The goal of the agent is to move forward as much as possible with a cheetah-like figure.
Hopper (version v2)	11	3	[-1,1]	The goal of the agent is to move forward as much as possible with a monoped figure.
Ant (version v2)	111	8	[-1,1]	The goal of the agent is to move forward as much as possible with an ant-like quadroped.

Table 5.1: Benchmark Environment descriptions.

5.2 Algorithms

We compare against several Q-learning algorithms designed for continuous actions, including QT-Opt, Wire-Fitting, NAF, PICNN, and Soft Q-learning. We also evaluate other algorithms that learn both value function and policy — namely Actor-Critic algorithms: DDPG, Actor-Critic, and Soft Actor-Critic. These Actor-Critic algorithms are mainly included as baselines, providing some preliminary insights into Actor-Expert compared to Actor-Critic. The main goal of these experiments is to compare the utility of Actor-Expert compared to other methods based on Q-learning.

For exploration, we use two different approaches, depending on whether the method has a stochastic exploration policy mechanism. QT-Opt, NAF, Actor-Critic, and Soft Actor-Critic have their own stochastic exploration mechanism.

For all other methods without a stochastic exploration policy mechanism like PICNN, Wire-fitting, Soft Q-learning, and DDPG, we add time-correlated OU noise (Lillicrap *et al.* 2016; Wawrzyński 2015) to the greedy action for exploration. OU noise is temporally correlated stochastic noise generated by Ornstein-Uhlenbeck process (Uhlenbeck *et al.* 1930). We set the parameters in the process to $\mu = 0.0, \theta = 0.15, \sigma = 0.2$.

5.2.1 Q-learning methods

QT-Opt (Kalashnikov *et al.* 2018): Unlike Actor-Expert, QT-Opt uses Cross Entropy Method to rather simply resolve the maximization from scratch at each step. Greedy action is obtained by executing two iterations of Cross Entropy Method with 64 samples on the action-values and fitting a Gaussian policy to the top 10%. This method provides a useful baseline to gauge if the Conditional CEM algorithm provides benefits, or if the main benefit is simply from using a CEM approach. The original work fits a unimodal Gaussian policy, but we also try fitting a bimodal Gaussian mixture policy to match Actor-Expert.

Wire-fitting (Baird and Klopf 1993; Gaskett *et al.* 1999): Action-values are restricted by the number of action control points. We use 100 action control points for the Bimodal Environment and 1000 for Benchmark domains. Greedy action is selected by finding the maximal action among the control points. OU noise is used for exploration instead of epsilon-greedy used in the original paper.

Normalized Advantage Function (NAF) (Gu, Lillicrap, Sutskever, *et al.* 2016): Action-values are restricted as a quadratic concave function. Maximal action $\mu(s)$ is implicitly found as a closed-form solution through this constrained action-value function. Exploratory action is selected by sampling from a Gaussian with the learned mean $\mu(s)$ and learned covariance $\Sigma(s)$, with initial exploration scale swept in $\{0.1, 0.3, 1.0\}$.

Partially Input Convex Neural Networks (PICNN) (Amos *et al.* 2017): Action-values are restricted to be concave with respect to actions. Without an explicit policy nor a closed-form solution for maximal action, the

greedy action is obtained by five iterations of bundle entropy method (approximate gradient ascent) from a randomly initialized action as suggested in the original paper. For exploration we use OU noise.

Soft Q-learning: This is an Actor-Expert-like method that optimizes a different entropy regularized objective. Greedy action is obtained by sampling from its policy which is proportional to its learned soft action-values. The action-values are learned via Soft optimal Bellman updates, and the temperature is subsumed into the reward according to the paper. Hence we sweep over reward scale factors of $\{1, 10, 100, 1000\}$. We use 30 samples for the empirical estimate of soft values and SVGD update, to match the sampling process in Actor-Expert. OU noise is used for exploration.

Actor-Expert/Actor-Expert+: We test two versions of Actor-Expert, Actor-Expert using the Quantile Empirical Distribution (Alg. 2) and Actor-Expert+ using the Optimized Quantile Empirical Distribution (Alg. 3). We use a bimodal Gaussian mixture for both Actors, with $N = 30$ and $\rho = 0.2$ for Actor-Expert and $N = 10$ and $\rho = 0.6$ for Actor-Expert+. For benchmark environments, it was even effective—and more efficient—for Actor-Expert+ to sample only 1 action ($N = 1$), with $\rho = 1.0$.

To select the maximal action in Actor-Expert using $\pi_{\mathbf{w}}$, we used the simple heuristic of selecting the mean with the highest coefficient in the Gaussian mixture. We found this choice was sufficient in these experiments, without needing to more carefully use $\pi_{\mathbf{w}}$ to find the maximum. We provide some empirical justification later in Chapter 9. Note that this choice only disadvantages Actor-Expert, and none of the competitors. Future experiments with Actor-Expert could investigate improved strategies for using $\pi_{\mathbf{w}}$ to find the maximal action.

5.2.2 Actor-Critic methods

Actor-Critic: We use the variant Actor-Critic update defined in Eq. (2.12), except by using $\mathbb{E}_\pi[Q^\pi(s, A)]$ as an estimate of $V_\theta^\pi(s)$ for the Actor update. We use a sampled average $\frac{1}{n}\sum_{i=1}^{n} Q^\pi(s, a_i)$ to approximate $\mathbb{E}_\pi[Q^\pi(s, A)]$, where a_i is sampled from the current policy $\pi_{\mathbf{w}}$, and $n = 30$.

Using the off-policy objective (Eq. (2.9)) also enables us to use experience replay to match the sample-efficiency of q-learning methods, though technically the policy gradient under off-policy sampling is no longer correct (Degris, White, *et al.* 2012). Action-values are updated using Sarsa[1] on transition $(S_t, A_t, R_{t+1}, S_{t+1})$, where action A_{t+1} is sampled from $\pi_{\mathbf{w}}(\cdot|S_{t+1})$ for the target in the update: $R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})$. The policy is updated by sampling only states S_t from the buffer, and computing the gradient using an action a sampled from the policy $\pi_{\mathbf{w}}(\cdot|s)$: $\nabla_{\mathbf{w}} \log \pi(a|s)[Q^\pi(s,a) - \frac{1}{n}\sum_{i=1}^{n} Q^\pi(s,a_i)]$. The exploratory action is taken by sampling from the policy.

Deep Deterministic Policy Gradient (DDPG)(Lillicrap *et al.* 2016): This Actor-Critic method learns a deterministic policy, and its output is selected as the greedy action. The policy is updated using a deterministic policy gradient theorem (Silver *et al.* 2014), and the action-values for this policy is learned by using $R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi(S_{t+1}))$ as target. OU noise is used for exploration.

Soft Actor-Critic(Haarnoja, Zhou, *et al.* 2018): Similar to Soft Q-learning (Haarnoja, Tang, *et al.* 2017) this method also optimizes an entropy regularized objective. But Soft Actor-Critic learns the action-values of the current policy, and instead of learning an approximate sampling network for the energy distribution, a Gaussian stochastic policy is learned by minimizing the KL divergence to the energy distribution. A Gaussian mixture policy could also be learned, but we follow the original paper where it chooses to learn the Gaussian policy to update the policy with a lower variance estimator using the reparameterization trick. Soft Actor-Critic learns a total of three networks: Q_π, V_π and π, where V_π is learned to stabilize learning. We however do not employ other techniques used to improve performance, such as learning two Q-functions. We subsume the temperature parameter into the reward, and sweep over reward scale factors of {1, 10, 100, 1000}. Like Actor-Critic, we also sample from the policy to take exploratory actions.

[1]Though typically used on-policy, Sarsa can be used for off-policy policy evaluation by simply sampling $A_{t+1} \sim \pi_{\mathbf{w}}(\cdot|S_{t+1})$ for the update, but not in the environment. See (Sutton and Barto 2018, Section 6.6) for more details.

5.3 Experimental Settings

Agent performance is evaluated every n steps of training, by executing the current policy without exploration for 10 episodes. We use offline performance evaluation, because Q-learning learns the optimal policy off-policy. The performance is averaged over 30 runs for the Bimodal Environment and Pendulum Environment, and 10 runs for other benchmark environments, with different fixed seeds for each run.

We sweep over learning rates – policy: {1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2}, and action-values: {1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1, 1e0}, and other relevant hyperparameters for each agent. For higher benchmark domains (HalfCheetah, Hopper, and Ant) we do a coarser sweep over learning rates – policy: {1e-3, 1e-4, 1e-5}, action-values: {1e-2, 1e-3, 1e-4}, as exhaustive parameter sweep is computationally expensive. Best hyperparameter settings for all agents are reported in Appendix A.

All agents use a neural network of two layers with 200 hidden units each, ReLU activations between each layer and tanh activation for action outputs. To allow fair comparison to Q-learning methods with similar number of parameters, the two networks in the Actor-Expert share the first layer, and branch out into two separate layers as described in the earlier chapter. When comparing to Actor-Critic methods with two separate networks, we also separate the networks for the Actor and the Expert to match the number of parameters.

All agents use an experience replay buffer and target networks, as is common when using neural networks for function approximation. We use a mini-batch size of 32, with buffer size = 10^6 and soft target network update(τ = 0.01).

5.4 Conclusion

In this chapter we described the environments and the algorithms to be evaluated in detail. In the following three chapters we will present our empirical results.

Chapter 6

Comparison in the Bimodal Environment

In this chapter, we compare algorithms in the toy Bimodal Environment introduced in the previous chapter. We first compare among Q-learning methods, and then provide results for algorithms learning both value-functions and policies.

6.1 Comparison of Q-learning methods

The Bimodal Environment has a single optimal action, so Q-learning should be able to find the optimal policy that selects this action. Despite this, a Q-learning agent could fail to find this optimal policy if its action-values are restrictive. Such action-values will be incapable of representing true action-values, and may either converge to a poor approximation that tries to balance error for these two regions or will continue to oscillate between the two regions. We hypothesize then, that the Q-learning agents that put such restrictions on the action-values will often fail in this environment.

We plot the average performance of the best setting for each agent over 30 runs, in Figure 6.1. Actor-Expert methods (Actor-Expert and Actor-Expert+) converged most reliably and quickly to the optimal policy, and having a bimodal policy seemed to perform slightly better than a unimodal policy. Actor-Expert+ performance was affected more when using a unimodal policy. All the methods that restrict the functional form for action-values failed in many

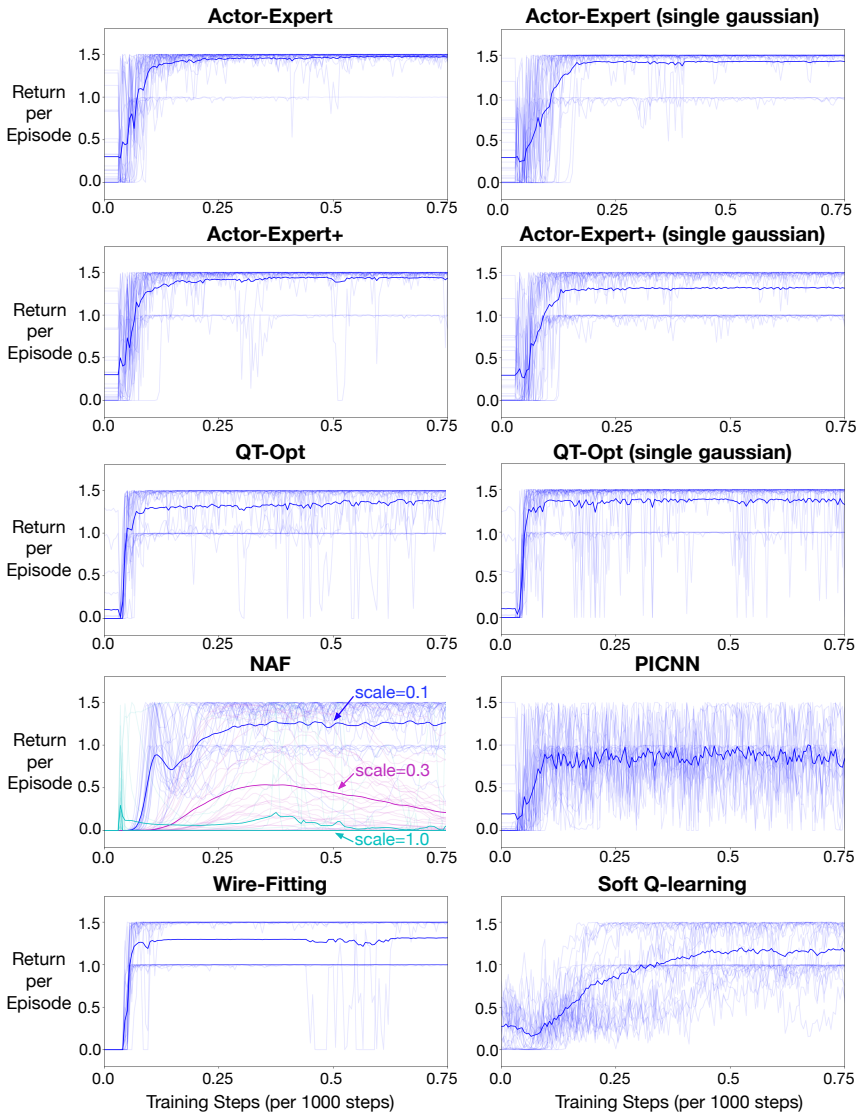

Figure 6.1: Q-learning methods evaluated on the Bimodal Environment. Each faded line represents one run while the dark line represents the average. Note that the restriction on action-value was critical, while Actor-Expert methods with bimodal policy performed well. As long as the action-values can be accurately learned, the policy can concentrate density around the single optimal action without being skewed by the suboptimal high-value region. QT-Opt and Wire-Fitting converged reliably to either optimal or suboptimal policy. Soft Q-learning was an exception however, and this may have been due to stochastic greeady action-selection process, proportional to its action-values.

runs. NAF and PICNN struggled to converge to neither the optimal nor suboptimal policy, resulting in much worse performance. When PICNN and NAF start to increase value for one action center, they necessarily have to decrease the action-values around the other action center because the action-values are

concave. Consequently, when the agents explore and observe higher reward for an action than they predict, it can skew the action-value estimates.

This was particularly visible for NAF with initial exploration scale=1.0. Due to large exploration, the agent sees both large rewards and tries to fit a large quadratic function encompassing both, resulting in the maximal action centered around action=0.0 with reward 0.0. For smaller exploration scale=0.1, the limited exploration allows the agent to converge more often to either of the action centers, sometimes finding the optimal action and sometimes finding the suboptimal action. Oscillation behavior between the two action centers was not observed, because the agent learns not through purely online samples but through experience batches from the replay buffer.

For PICNN, the agent did not seem to robustly converge to either optimal or suboptimal actions even though it has a more general functional form than NAF. From our own visual observation of the videos of the learned action-values over time, the action-values do manage to center around the optimal action for some runs. But even in these ideal runs, the action-values were quite pointed and as a result, the optimization over actions oscillated around the maximal action. It is nonetheless clear that PICNN suffers from restricted action-values, because in about half of the runs, it settled around the suboptimal action.

Soft Q-learning is capable of learning correct soft action-values but it failed to stably converge to a good policy. Soft Q-learning learns a sampling network that learns to sample actions proportional to the learned action-value energy distribution. Actions sampled from this sampling network is taken as the maximizing action. Thus, the greedy action-selection is stochastic and also depends on whether there are suboptimal actions that also have high action-values. Furthermore, on certain runs we observed only a single peak in the learned soft action-value function. We hypothesize this is due to using a limited exploration strategy of adding OU noise to the greedy action, making it more likely to miss a better state-action.

Wire-Fitting and QT-Opt were also capable of correctly modeling action-values, but both surprisingly converged to the suboptimal policy quite often.

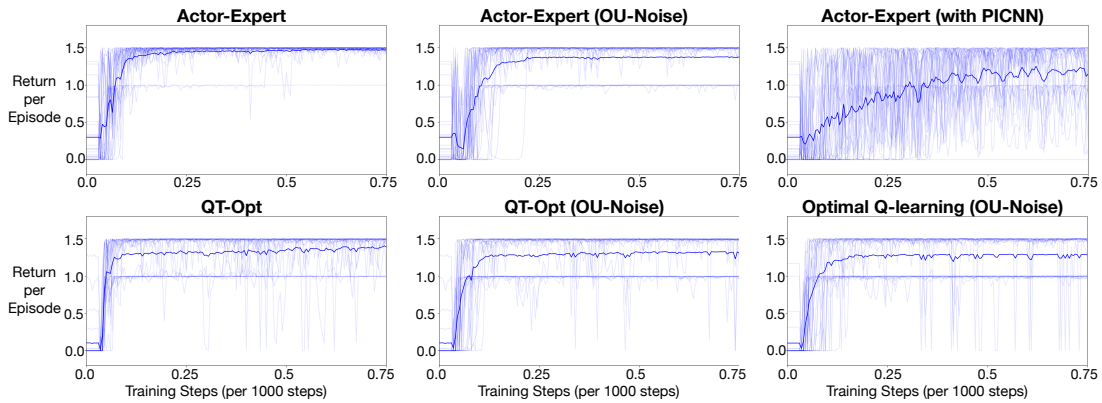

Figure 6.2: Evaluating the effects of exploration on the Bimodal Environment. Performance degradation is observed for using OU noise instead of the stochastic policy for exploration, but it is minor compared to the effect of restricting the action-values.

This may also be due to the lack of efficient exploration mechanism, particularly using OU noise rather than having an explicit stochastic Actor in Wire-Fitting or learning an Actor per-step using CEM, rather than across steps in QT-Opt. To test this hypothesis, we run additional experiments to better separate the effects of restricting the functional form of the action-values and the type of exploration used.

In Figure 6.2, we test variants of Actor-Expert, using OU noise or combining with restricted action-values. We also include QT-Opt with OU noise and an idealized Q-learning agent (Optimal Q-learning), where we discretize the action space finely (in increments of 0.001) and evaluated all action values to find the exact maximal action-value/action at each time step. The action-value function is parameterized by a neural network, with architecture identical to other methods. This idealized agent removes confounding factors in restricting actions and in finding optimal actions, and just reflects any issues with OU noise exploration.

The performance is worse when Actor-Expert uses OU noise for exploration, but for the majority of runs it still converges to the optimal policy. Once we use a restricted action-value for Actor-Expert, by using the same network architecture as PICNN for the action-values, then performance becomes almost as bad as PICNN. Furthermore, even the ideal Optimal Q-learning

failed to learn the optimal policy for all runs, due to local exploration with OU-noise.

There are several other interesting points, in addition to this main outcome. Actor-Expert outperforms QT-Opt, either when selecting actions from the stochastic actor or when using OU-noise for exploration. This validates the utility of learning the maximizer with CCEM, rather than re-optimizing with CEM on each step.

This experiment has shown that having a stochastic policy for exploration is helpful. We perform further analysis into this when we compare Actor-Expert methods with other methods that learn both value function and policy in the later sections.

So far these results overall suggest that it is both useful to have a more general form for the action-values and an explicit Actor for exploration.

6.2 Comparison of Methods that Learn Both a Value-function and a Policy

In this section, we provide a comparison between Actor-Expert methods and Actor-Critic methods. These methods all learn both a value-function and a policy, and can shed light on the differences between Actor-Expert and Actor-Critic methods. As mentioned earlier, for this comparison we do not use the branching architecture of Actor-Expert and use separate networks for the Actor and the Expert to keep the number of parameters as similar as possible.

In Figure 6.3 we observe that Actor-Expert methods converge more often to optimal action-values while other methods do not. Methods with a stochastic action-selection during evaluation (Actor-Critic, Soft Actor-Critic, and Soft Q-learning) seemed to be especially noisy and have worse performance than methods with deterministic action-selection during evaluation. DDPG has a deterministic policy and converged stably, to either the optimal action or suboptimal action.

Actor-Critic performed worse than Actor-Expert and was noisy. The performance seemed to be particularly sensitive to the bounds of its variance.

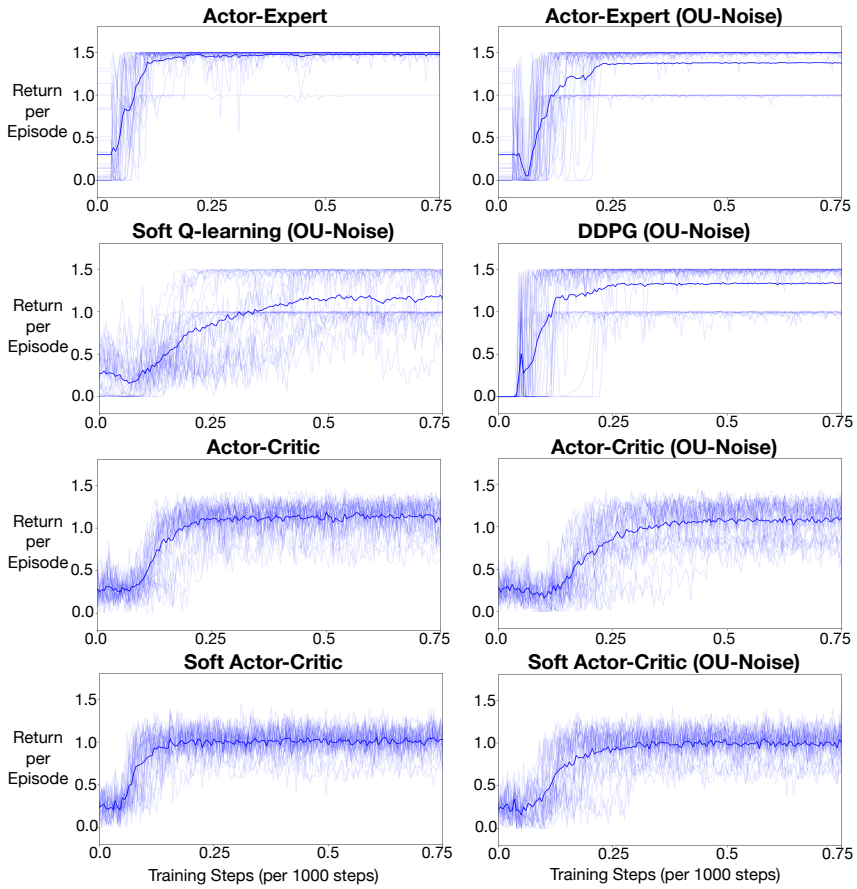

Figure 6.3: Evaluating methods that learn both value-function and policy on the Bimodal Environment. Actor-Expert methods converge robustly while other methods do less so. Methods that use a deterministic action-selection during evaluation seem to do better than those that use stochastic action-selection.

Even when the action-value is peaked around the optimal action and the policy correctly tracks it, during evaluation the action sampled from the policy can be far from the optimal action, leading to an average reward of less than 1.5. For Actor-Critic and Soft Actor-Critic in this Bimodal Environment we allowed a lower variance bound with a wider range: $[e^{-4}, e^{4}]$ in order to obtain better performance. There was little difference in using unimodal or bimodal policy for Actor-Critic and Soft Actor-Critic.

6.3 Hyperparameter Sensitivity to Learning Rates

In this section, we provide analysis of sensitivity to learning rate parameter for each methods. Hyperparameter sensitivity curve shows how robust a method is to changes in hyperparameters. For the range of learning rate that we sweeped over, we select other hyperparameters with the best result. In Figure 6.4 we plot sensitivity curve of action-value learning rate for Q-learning methods. Actor-Expert methods, QT-OPT, and OptimalQ do not have pointed peaks, indicating they are less sensitive and perform equally well for a larger range of action-value learning rates. Other methods like WireFitting, NAF, Soft Q-learning, and PICNN seemed more sensitive to action-value learning rates.

In Figure 6.5 we plot action-value and policy learning rates for Actor-Expert and Actor-Critic methods. Most methods seemed less sensitive to action-value learning rates. For policy learning rates, most methods preferred learning rates between range $5e-4$ and $5e-3$, while Soft Q-learning performed the best with a low policy learning rate.

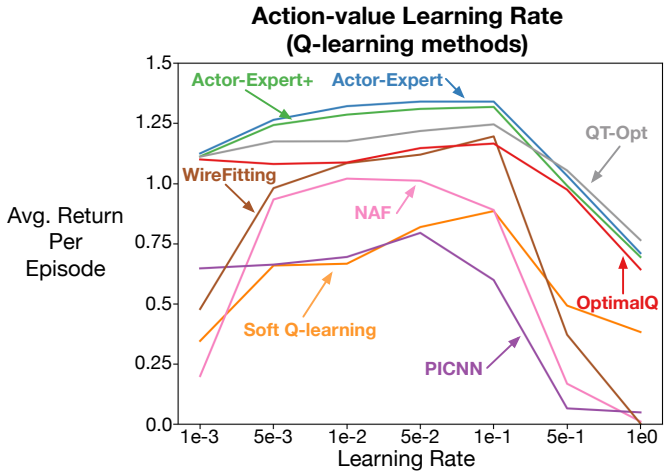

Figure 6.4: Sensitivity curve for action-value learning rates. Actor-Expert methods, QT-Opt, and OptimalQ seemed less sensitive to action-value learning rates.

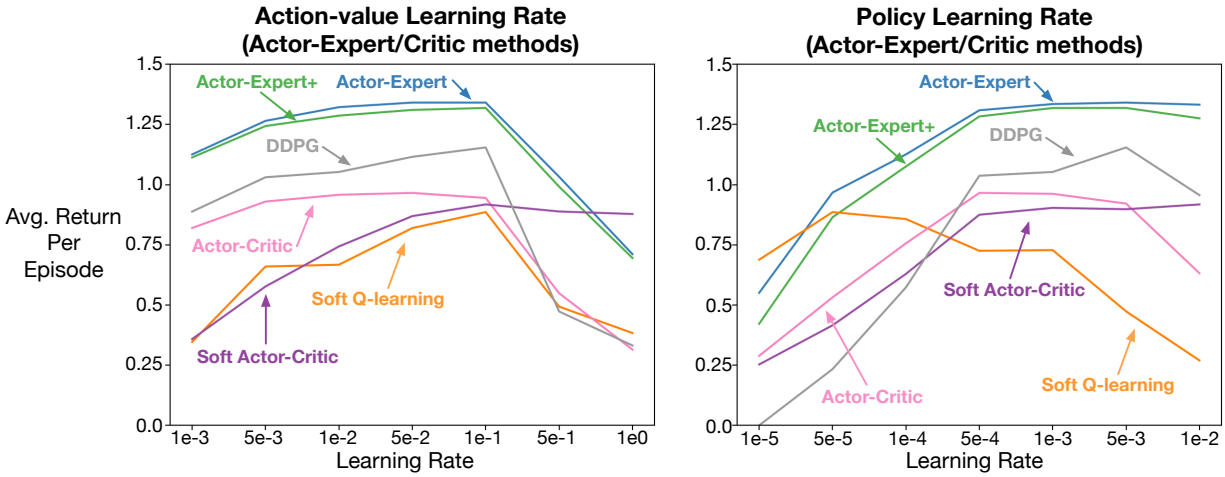

Figure 6.5: Sensitivity curve for action-value and policy learning rates. Soft Q-learning prefers much smaller policy learning rate than other methods.

6.4 Conclusion

In this chapter, we presented experimental results in the Bimodal Environemnt. We found that restricting action-value functions can be detrimental and often lead to suboptimal performance. Having a stochastic Actor policy also helped with exploration, which many previous Q-learning methods lack. Actor-Expert however, does not constrain the action-value function and has a stochastic Actor policy that allowed it to almost always learn the optimal policy. Furthermore, Actor-Expert converged more reliably to the optimal policy than QT-Opt, validating that Conditional CEM seems more effective as well as being efficient than performing CEM at each step. We also provided sensitivity analysis on the learning rates, validating the sweep range as well as looking at how sensitive each method was to its learning rates.

Chapter 7

Comparison in the Pendulum Environment

In this chapter we move from the previous bandit setting and compare algorithms in a reinforcement learning setting on the Pendulum environment. Each episode terminates after 200 steps. We perform the same hyperparameter sweeps and 30 runs. The results are smoothed over a moving window average of size 10.

7.1 Comparison of Q-learning methods

Comparison of Q-learning algorithms is shown in Figure 7.1. Wire-fitting is omitted from these plots, as it performed poorly on Pendulum.

Most Q-learning algorithms performed well in this domain, except for PICNN which seemed to converge to a suboptimal policy and Soft Q-learning which failed to learn a good policy.

NAF and PICNN which restrict action-value function performed well, which might indicate that the true action-value function happen to match this parameterization and is unimodal. We hypothesize that Soft Q-learning did not perform well because this environment requires precision and stochasticity is harmful.

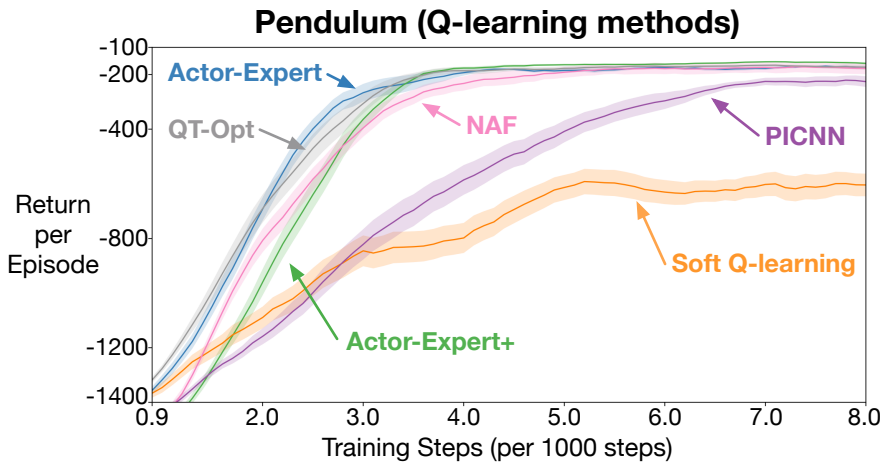

Figure 7.1: Comparison of Q-learning methods and Actor-Expert evaluated on Pendulum environment. Results are averaged over 30 runs and smoothed over a moving window average of size 10. The shaded region represents standard error.

7.2 Comparison of Methods that Learn Both Value-function and Policy

Next we provide a comparison between Actor-Expert methods and Actor-Critic methods, methods that learn both value-function and policy, shown in Figure 7.2.

All Actor-Critic methods (DDPG, Actor-Critic, and Soft Actor-Critic) do not reach the same level of performance as Actor-Expert methods by the end of training. Furthermore they tend to exhibit larger standard error than Actor-Expert methods.

Interestingly, Soft Q-learning and Soft Actor-Critic performed similarly worse, and we again believe it is due to their preference of stochasticity when optimizing an entropy-regularized objective.

7.3 Hyperparameter Sensitivity to Learning Rates

In this section we provide similar hyperparameter sensitivity analysis in Pendulum environment. Figure 7.3 shows the sensitivity curve for action-value learning rates in Q-learning methods. All methods except Soft Q-learning

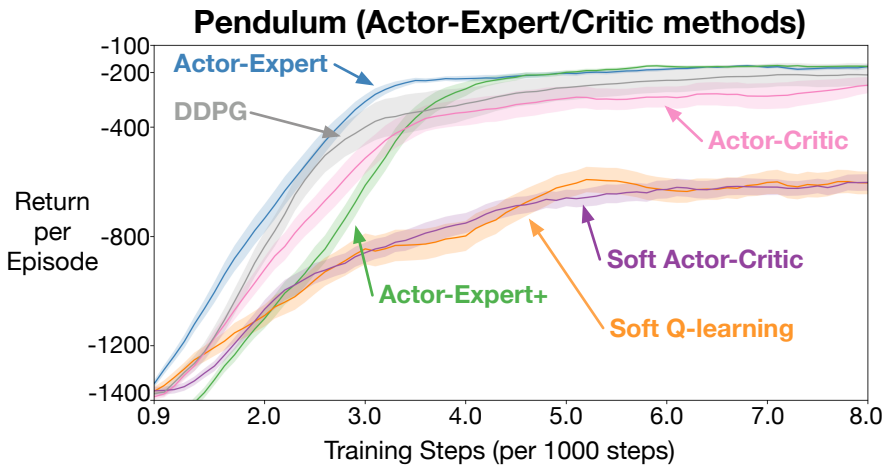

Figure 7.2: Comparison of Actor-Expert methods with Actor-Critic methods on Pendulum environment, smoothed over a moving window average of size 10. The shaded region represents standard error.

showed similar sensitivity. For NAF, better performance may have been obtained by using an action-value learning rate lower than $1e-3$.

Figure 7.4 shows the sensitivity curve for action-value and policy learning rates in Actor-Expert and Actor-Critic methods. Both Soft Q-learning and Soft Actor-Critic showed different sensitivity curve from rest of the methods and seemed to be less sensitive to learning rate hyperparameters, but then their performances were quite worse compared to other methods.

Compared to sensitivity curve in Bimodal Environment, all methods seemed to favor lower learning rates, likely due to higher dimensional state space and environment complexity.

7.4 Conclusion

In this chapter we presented results on Pendulum environment, extending our comparison of methods from the bandit setting to a reinforcement learning setting. Actor-Expert methods performed well compared to all other methods, especially compared to Actor-Critic methods. Q-learning methods mostly converged to the optimal policy, even methods that also restricted the action-value function. The true action-value function may have been unimodal and easy to solve; in the next chapter we run experiments on even higher-dimensional

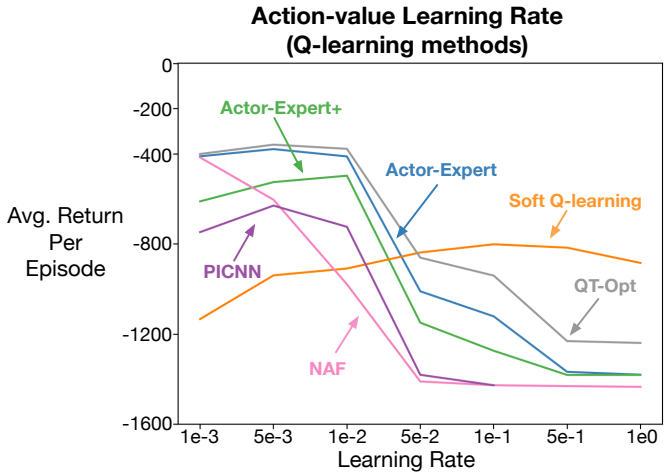

Figure 7.3: Sensitivity curve for action-value learning rates in Q-learning methods.

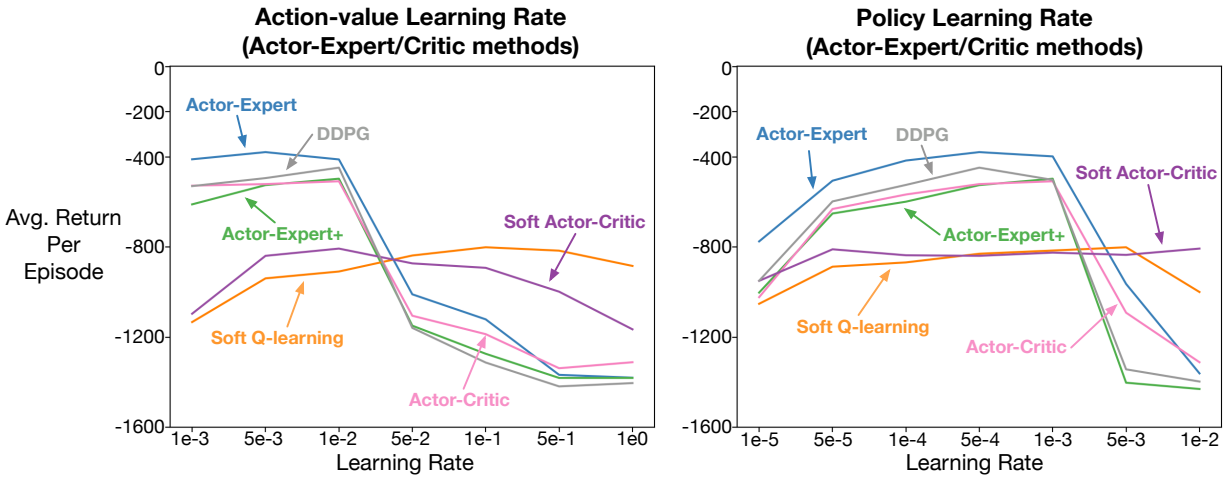

Figure 7.4: Sensitivity curve for action-value and policy learning rates in Actor-Expert/Critic methods.

benchmark environments to further validate Actor-Expert methods.

Chapter 8

Demonstration in Benchmark Environments

In this chapter we compare algorithms in three additional higher-dimensional continuous action space environments. For these domains, hyperparameter sweeps and runs become more computationally expensive. In HalfCheetah, Hopper, and Ant environment we do coarser learning rate sweeps with 10 runs. Similar to Pendulum the results are smoothed over a moving window average of size 10. We also include previous results in Pendulum in the plots for better overview. The results for these three additional higher dimensional environments serves to provide a sanity check that Actor-Expert scales and suggest its competitiveness.

8.1 Comparison of Q-learning methods

We evaluate Q-learning algorithms, with results shown in Figure 8.1. Actor-Expert methods (Actor-Expert and Actor-Expert+) perform consistently well across all environments. Other methods that perform well in one environment does not perform as well in other environments.

NAF and PICNN have the poorest performance. NAF converges to a good policy only in Pendulum, and performs poorly in other environments and especially suffers in Hopper and Ant. PICNN learns slowly and seems to learna suboptimal policy in Pendulum, and otherwise is generally only better than NAF and similar to QT-Opt.

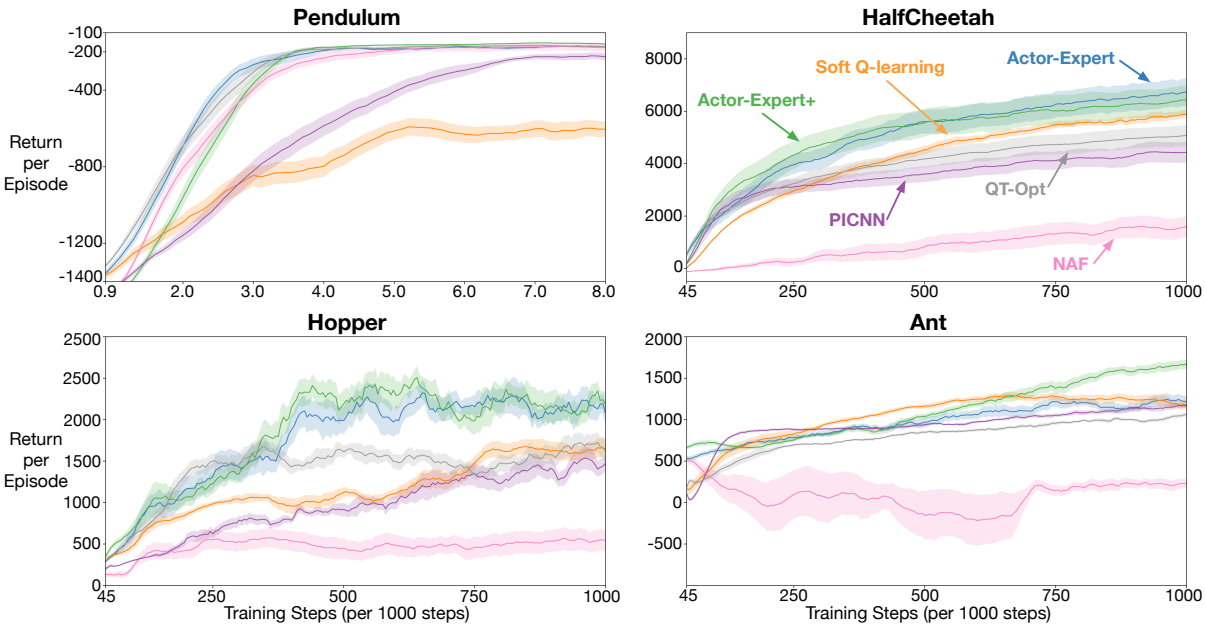

Figure 8.1: Comparison of Q-learning methods and Actor-Expert evaluated on benchmark environments. Results are smoothed over a moving window average of size 10. The shaded region represents standard error.

Of note is that we do not see the same instability in PICNN as we saw in Bimodal environment. This may be because for locomotion environments like HalfCheetah, Hopper, and Ant, precision is not necessary and selecting approximately good actions is enough to achieve reasonable performance, or because the action-values were less pointed for these environments.

Soft Q-learning although suffering in Pendulum, does surprisingly well in other environments. This could also be because locomotion environments do not require precision and stochasticity induced by the entropy term can even be desirable to learn a more robust policy.

Actor-Expert methods also consistently outperform or match QT-Opt, again suggesting that using CCEM to learn a maximizer with CEM is more effective than simply using CEM to approximate the maximal action on each step. This provides some evidence that it is effective to learn a maximizer—an Actor—using CCEM. It is possible that QT-Opt would match, or even outperform Actor-Expert given more optimization iterations of CEM in QT-Opt. This would, however, significantly increase the computational complexity as compared to Actor-Expert and likely be prohibitively expensive in higher di-

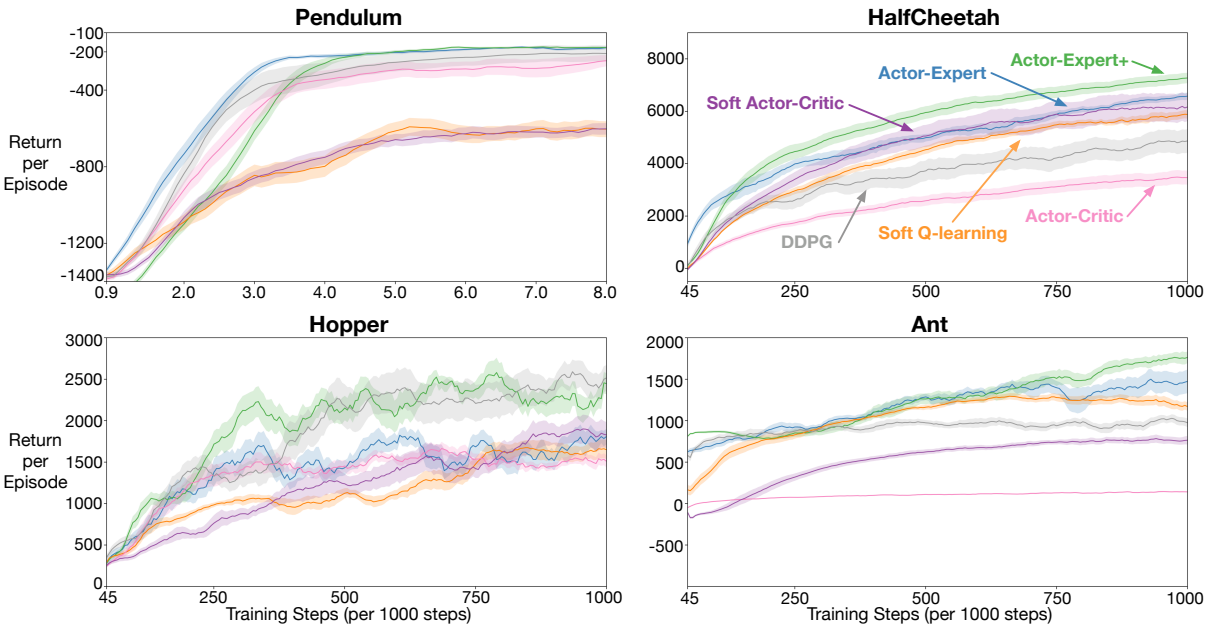

Figure 8.2: Comparison of Actor-Expert methods with Actor-Critic methods on benchmark environments, smoothed over a moving window average of size 10. The shaded region represents standard error.

mensional environments.

8.2 Comparison of Methods that Learn Both Value-function and Policy

We also provide a comparison between Actor-Expert methods and Actor-Critic methods, methods that learn both value-function and policy, shown in Figure 8.2.

Soft Q-learning and Soft Actor-Critic perform well in HalfCheetah but not as much in other environments. They especially suffer in Pendulum which we hypothesize is due their preference of stochasticity when the environment requires precision.

Actor-Critic performs well only in Pendulum, while failing to learn as good policy in other domains. We hypothesize that one of the reasons Actor-Expert outperforms Actor-Critic is that during evaluation Actor-Critic selects action sampled from its stochastic policy, whereas Actor-Expert selects a deterministic greedy action (of taking the mean with the higher coefficient). This suggests

that a stochastic Actor for exploration is useful but that a deterministic policy during execution can obtain better performance, which was also observed in the previous bimodal environment. This is further supported by DDPG outperforming Actor-Critic in all environments.

8.3 Conclusion

In this chapter we presented experimental results on benchmark environments, suggesting the competitiveness of Actor-Expert compared to other methods. Compared to previous Q-learning methods Actor-Expert consistently performs well in all environments. Compared to other methods with both value-function and policy similar conclusion was drawn, with additional insight that using a deterministic policy for evaluation seems to perform better than methods with a stochastic policy. The results suggest that the Actor-Expert framework can be valuable in exploring approximate Q-learning methods in continuous action spaces. However, further investigation is still required to discern the differences in learning Q_* and Q_π.

Chapter 9

Investigating Particular Choices in Actor-Expert

In this chapter, we provide more experimental analysis of Actor-Expert methods. We first justify our choice of using the mean of the Gaussian as a heuristic to estimate the maximizing action. Then we compare the effect of sharing the representation layer. Lastly, we explore ways to improve Actor-Expert by augmenting the CEM update in the Actor.

9.1 Using the Mean of the Gaussian as the Estimate for Maximizing Action

Here we provide justification on using the mean of the Gaussian with the higher mixing coefficient as a heuristic for obtaining the maximizing action in Actor-Expert. We compare our results on Pendulum and HalfCheetah. Maximizing action-value and its corresponding action is needed in two occasions: during Q-learning update and during evaluation. To assess whether using the mean of the Gaussian with the higher mixing coefficient is sufficient, we performed maximum 10 gradient ascent steps from the mean to find a better action-value target and its action. The result is shown in Figure 9.1.

We strongly controlled the seeds during the run, and thus there is little randomness in the transitions except through different action-selection by the agent. For both environments, the error bars overlap and there is no significant difference between using gradient ascent and not using gradient ascent during

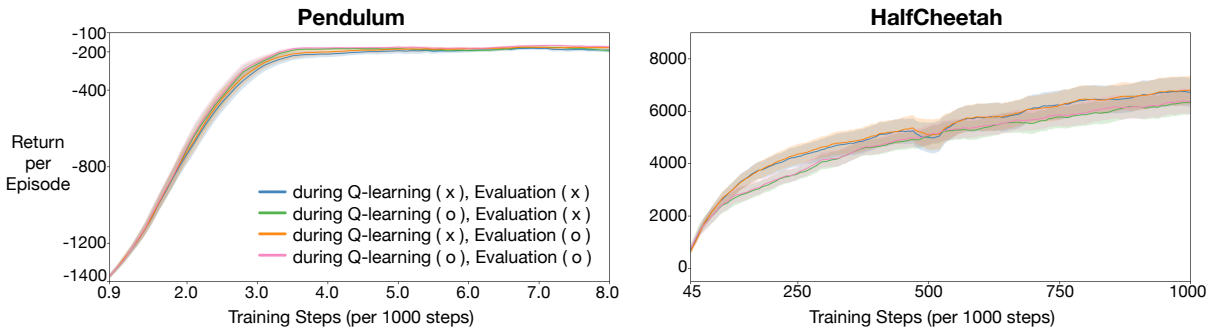

Figure 9.1: Actor-Expert with better maximizing action and target action-value. We compare the effect of using a better maximizing action/action-value by performing gradient ascent during the Q-learning update and evaluation.

the Q-learning update or evaluation. Overall, we can only conclude that there is little gain when performing expensive gradient ascents to obtain a better maximizing action. Simply taking the mean can be a viable alternative to obtaining a reasonable maximizing action.

In our previous experiments with methods that learn both action-values and a policy, we had observed that methods with a stochastic action-selection for evaluation(Actor-Critic, Soft Actor-Critic, Soft Q-learning) perform worse than methods with a deterministic action-selection (Actor-Expert, DDPG). To test this further, we evaluate Actor-Critic and Soft Actor-Critic by taking the mean of the parameterized Gaussian policy, serving as an estimate for the maximizing action similar to Actor-Expert. For Actor-Critic we use bimodal Gaussian mixture to match Actor-Expert so we take the mean with the higher coefficient. The comparison is shown in Figure 9.2 and Figure 9.3. By using the mean, Actor-Critic and Soft Actor-Critic perform much better, similar and sometimes even better than Actor-Expert methods.

In Actor-Critic methods, evaluating actions should normally be selected by sampling from the current policy as the action-value functions are learned with respect to this policy. By taking the mean, the methods resemble Actor-Expert more where the Actor merely helps to find the maximizing action. If the action-values were to be learned from this maximizing action instead of from the sampled actions it would become an Actor-Expert method.

The results in this section suggests that using the mean of the Gaussian pol-

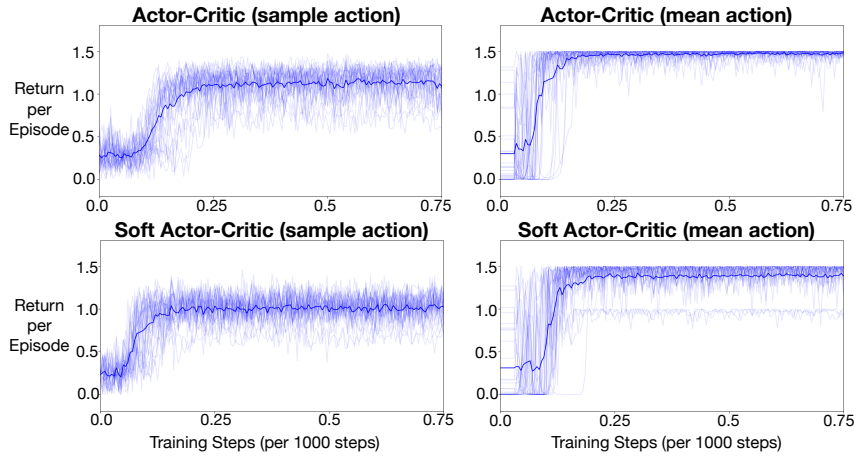

Figure 9.2: Comparison of Actor-Critic and Soft Actor-Critic taking samples or taking the mean from the parameterized policy in Bimodal Environment during evaluation.

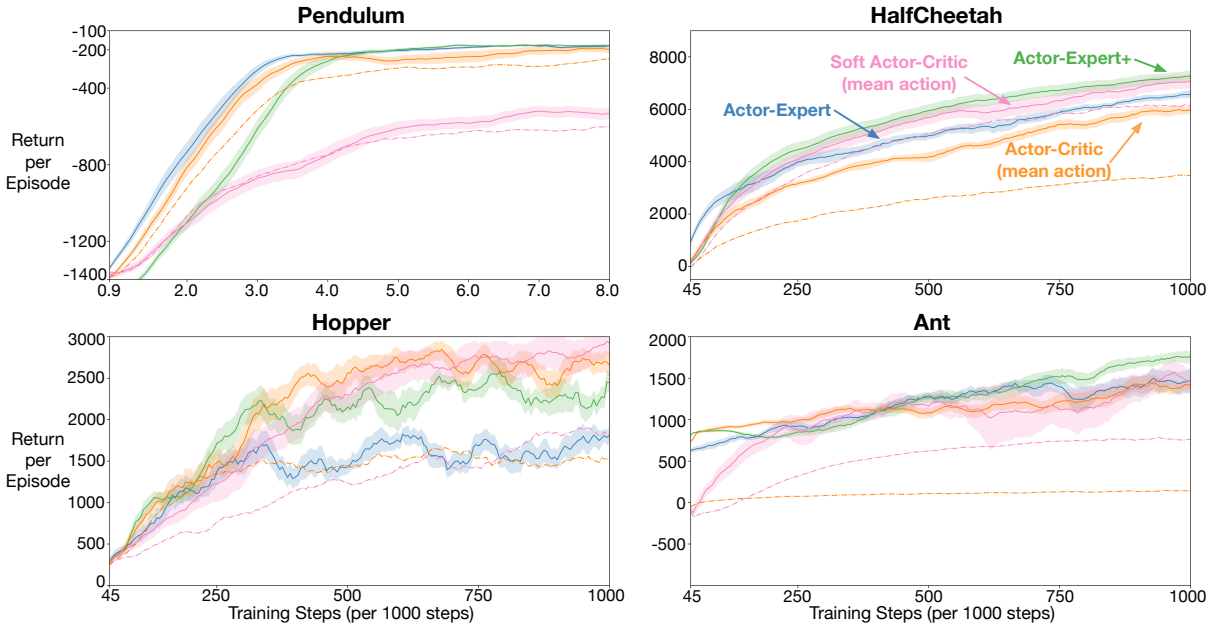

Figure 9.3: Comparison of Actor-Expert methods with Actor-Critic methods (taking the mean during evaluation) on benchmark environments. The dotted lines represent previous results of taking samples from the policy during evaluation.

icy serves as a reasonable estimate of the maximizing action. Using the mean for evaluation in Actor-Critic methods also greatly improved performance, suggesting the benefit of using the Actor to aid in finding the maximizing action rather than using the Actor directly.

9.2 Effect of Sharing the Representation Layer

In previous experiments we had modified the architecture of Actor-Expert methods to match the number of parameters when comparing against Q-learning methods and Actor-Critic methods. In this section we check if there is a noticeable performance difference due to changes in the architecture. In Figure 9.4, sharing the first layer seems to perform better in Pendulum, HalfCheetah, and Hopper while having a separate layer seems to perform better in Ant. All in all, there does not seem to be a significant performance difference between the two architectures used.

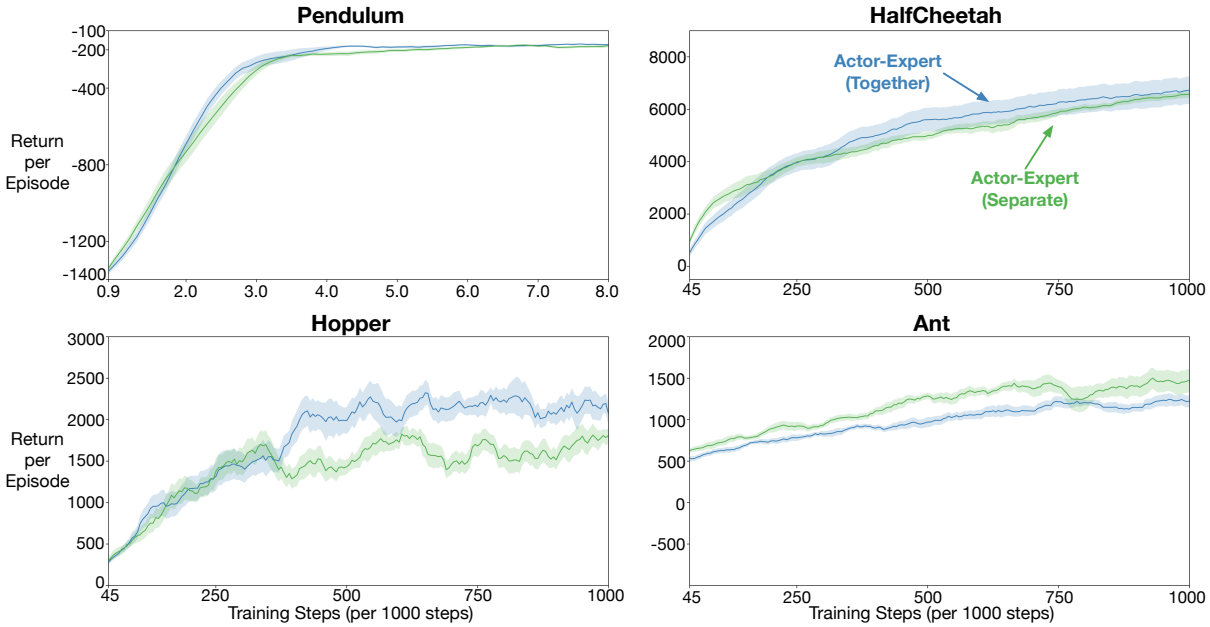

Figure 9.4: Actor-Expert sharing the representation layer or having a separate layer.

9.3 Adding Uniform Sampling to the Cross Entropy Method

Up to now, when using CEM, we have been using samples from the Actor. CEM is a global optimizer that converges robustly to an optimum if there is a non-zero probability of selecting each action in the action space. However in practice, we parameterize the Actor with a Gaussian mixture, where the

covariance matrix may not be large enough initially or shrink too quickly. To mitigate this effect and help the Actor find the maximizing action more robustly, we substitute 20% of the samples with uniform samples. The result is shown in Figure 9.5, and we see some performance gains in Pendulum and HalfCheetah, but not as much in Hopper and Ant. For other environments, the total number of samples and the augmented uniform samples may not have been enough to see the effect due to their high dimensionality.

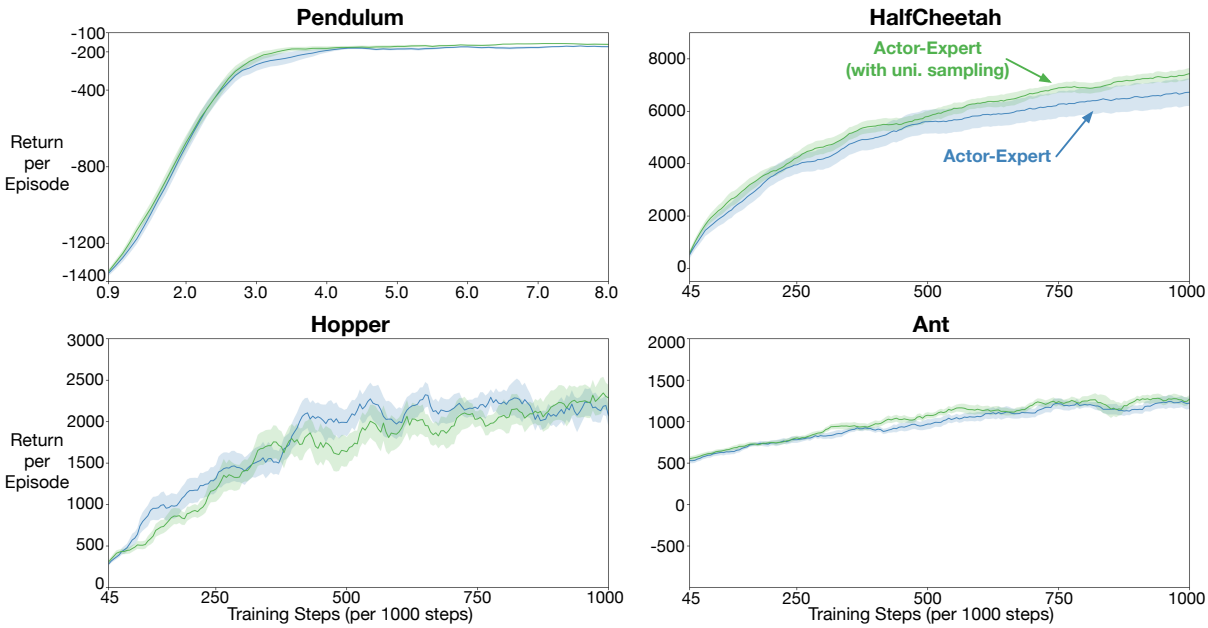

Figure 9.5: Actor-Expert augmented with uniform samples during the Cross Entropy Method update.

9.4 Conclusion

In this chapter we investigated particular design choices in Actor-Expert and analyzed its effects on performance. Using the mean of the Gaussian policy seemed to be a reasonable alternative to approximate the maximizing action. Sharing or separating the first layer also seemed to have similar performance. Lastly, we found that adding uniform random samples in CEM provided small, but mostly negligible gains. The results from this section validate our design choices in implementing Actor-Expert.

Chapter 10

Conclusion and Future Works

In this thesis, we have introduced a new framework called Actor-Expert, to facilitate the use of Q-learning in continuous action spaces. Value-based methods such as Q-learning, have been traditionally difficult to apply to continuous action spaces due to the hard optimization problem. Q-learning however offers some advantages over policy-based methods such as high sample-efficiency and use of off-policy samples. Previous value-based approaches for continuous control have typically restricted the action-value form, discretized the action space or used a different entropy-regularized reinforcement learning objective to make the maximization over actions easier.

Actor-Expert presents a different solution by decoupling action-selection from action-value representation. We introduce an Actor that optimizes a new policy objective, learning to identify maximal actions for the Expert action-values. This facilitates the use of Q-learning in continuous control, as the agent can quickly select actions without having to solve a potentially expensive optimization problem. The learned action-values are also decoupled from the learned policy, allowing the agent to learn optimal action-values, less reliant on the policy. We have also developed a practical instance of the Actor-Expert framework, by introducing a Conditional Cross Entropy Method to iteratively find greedy actions conditioned on states.

We then showed from empirical experiments that it can be problematic to restrict the action-value function, with failure from Q-learning methods using constrained action-value function to learn non-unimodal action-values. We

also found that having a stochastic policy Actor helped with exploration, that many previous Q-learning methods lacked. Actor-Expert methods are able to robustly find the optimal policy even when the true action-value function is bimodal, and perform as well as or better than previous Q-learning methods in several benchmark environments.

Overall, our results provide evidence that Actor-Expert with CCEM is a promising strategy for using Q-learning in continuous action spaces. It can learn to select greedy actions for multimodal action-values and provides a more effective exploration mechanism by sampling from its resulting distribution without any external exploration parameters to tune.

We also compared against methods that learn both the value function and policy. Most of the known methods belong to a class of policy-gradient methods called Actor-Critic methods, and we showed that Actor-Expert performs comparably. During evaluation, modifying the Actor-Critic methods to take a deterministic action by using the mean as the approximate maximizing action improved performance as well.

Like the Actor-Critic framework, we hope for the Actor-Expert framework to facilitate further development in optimizing our proposed policy objective. The separation into an Actor and a Critic enabled the two components to be optimized in a variety of ways, facilitating algorithm development. Actor can incorporate different update mechanisms to achieve better sample efficiency (Kakade 2001; Mnih *et al.* 2016; Peters *et al.* 2008; Wu *et al.* 2017) or stable learning (Schulman, Levine, *et al.* 2015; Schulman, Wolski, *et al.* 2017). Critic can be used as a baseline or control variate to reduce variance (Greensmith *et al.* 2004; Gu, Lillicrap, Ghahramani, *et al.* 2017; Schulman, Moritz, *et al.* 2016), and improve sample efficiency by incorporating off-policy samples (Degris, White, *et al.* 2012; Lillicrap *et al.* 2016; Silver *et al.* 2014; Wang *et al.* 2017). Preliminary comparison between Actor-Critic methods suggested that with more improvements Actor-Expert methods can outperform Actor-Critic methods. We hope our framework can spur more research into development of better Actor-Expert algorithms and exploring further benefits of Q-learning in continuous action spaces.

Our Actor-Expert framework also suggests that the key distinction for control algorithms is learning Q^* or Q_π. Typically, a separation is made between value-based (Q-learning) methods and policy-based methods. This separation however, becomes grey, particularly with methods like Actor-Critic that use both parameterized policies and parameterized values. Introducing a parameterized policy (Actor) into Q-learning methods provides another point in a spectrum between value-based and policy search methods, further undermining this distinction.

Instead, the new policy objective for the Actor in Actor-Expert highlights that a potentially more important distinction between control algorithms is whether the agent estimates the action-values of the current policy (Q_π) or the action-values of the optimal policy (Q^*) — decoupled from the policy. Since we can only do approximate Q-learning, we are technically not learning the action-values of the optimal policy, but a policy in between. Any estimate of the action-values — as long as it is decoupled from the policy — is in fact sufficient to fit into our framework. This distinction has of course featured prominently when distinguishing Q-learning and Sarsa wihin value-based methods; here we suggest this remains the key distinction even when moving to learning explicit policies. In our current approach we provide a heuristic in using the mean with the highest coefficient to approximate the maximal action.

In our future work, within the Actor-Expert framework, we hope to explore better alternatives to approximate the maximizing action, and different ways to update the Actor. In our instance of Actor-Expert, one limitation is that the quality of performance is closely related to the number of samples used in the updates. As the dimensions scale, there would be more computational burden to perform the Actor updates. Fortunately, with the introduction of a separate Actor, the action-selection process is unaffected and is still efficient during evaluation.

Actor-Expert framework also suggested that the new distinction is in learning a decoupled action-value function, similar to Q-learning and Sarsa. Exploring how Expected Sarsa fits into the spectrum could also be an interesting direction to explore.

References

[1] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *International Conference on Machine Learning*, 2017. 12, 29

[2] L. C. Baird, "Advantage updating," Wright Laboratory, Tech. Rep., 1993. 11

[3] L. C. Baird and A. H. Klopf, "Reinforcement learning with high-dimensional, continuous actions," Wright Laboratory, Tech. Rep., 1993. 11, 29

[4] S. Bhatnagar, M. Ghavamzadeh, M. Lee, and R. S. Sutton, "Incremental natural actor-critic algorithms," in *Advances in Neural Information Processing Systems*, 2008. 2

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016. 27

[6] M. Á. Carreira-Perpiñán, "Mode-finding for mixtures of Gaussian distributions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000. 18

[7] R. H. Crites and A. G. Barto, "An actor/critic algorithm that is equivalent to Q-learning," in *Advances in Neural Information Processing Systems*, 1995. 17

[8] T. Degris, P. Pilarski, and R. Sutton, "Model-free reinforcement learning with continuous action in practice," in *American Control Conference*, 2012. 1, 2, 9

[9] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," in *International Conference on Machine Learning*, 2012. 2, 8, 31, 55

[10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016. 2

[11] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," in *Advanced Topics in Artificial Intelligence*, 1999. 11, 29

[12] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, 2004. 55

[13] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," in *International Conference on Learning Representations*, 2017. 55

[14] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *International Conference on Machine Larning*, 2016. 11, 29

[15] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *International Conference on Machine Learning*, 2017. 13, 24, 31

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018. 24, 31

[17] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine learning*, 2011. 25

[18] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, 2003. 1, 19

[19] M. E. Harmon and L. C. Baird, "Multi-player residual advantage learning with general function approximation," Wright Laboratory, Tech. Rep., 1996. 12

[20] ——, "Residual advantage learning applied to a differential game," in *International Conference on Neural Networks*, 1996. 12

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Association for the Advancement of Artificial Intelligence*, 2017. 2

[22] J. Hu, P. Hu, and H. S. Chang, "A stochastic approximation framework for a class of randomized optimization algorithms," *IEEE Transactions on Automatic Control*, 2012. 19

[23] E. Imani, E. Graves, and M. White, "An off-policy policy gradient theorem using emphatic weightings," in *Advances in Neural Information Processing Systems*, 2018. 2

[24] S. Kakade, "A natural policy gradient," in *Advances in Neural Information Processing Systems*, 2001. 55

[25] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, 2018. 10, 21, 29

[26] H. Kimura, "Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and Gibbs sampling," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007. 10

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016. 2, 29, 31, 55

[28] S. Lim, A. Joseph, L. Le, Y. Pan, and M. White, "Actor-expert: A framework for using action-value methods in continuous action spaces," *arXiv preprint arXiv:1810.09103*, 2018. 22

[29] H. R. Maei, "Convergent actor-critic algorithms under off-policy training and function approximation," *arXiv preprint arXiv:1802.07842*, 2018. 2

[30] S. Mannor, R. Rubinstein, and Y. Gat, "The cross entropy method for fast policy search," in *International Conference on Machine Learning*, 2003. 1, 20

[31] L. Metz, J. Ibarz, N. Jaitly, and J. Davidson, "Discrete Sequential Prediction of Continuous Actions for Deep RL," *arXiv preprint arXiv:1705.05035*, 2017. 10

[32] J. d. R. Millán, D. Posenato, and E. Dedieu, "Continuous-action Q-learning," *Machine Learning*, 2002. 10

[33] P. Millington, "Associative reinforcement learning for optimal control," M.S. Thesis, Massachusetts Institute of Technology, Tech. Rep., 1991. 11

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016. 2, 55

[35] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *International Conference on Machine Learning*, 2007. 1, 19

[36] ——, "Natural actor-critic," *Neurocomputing*, 2008. 55

[37] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology And Computing In Applied Probability*, 1999. 3, 11, 15, 19

[38] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017. 19

[39] J. Santamaría, R. Sutton, and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive Behavior 6(2)*, 1998. 10

[40] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015. 55

[41] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *International Conference on Learning Representations*, 2016. 9, 55

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. 55

[43] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014. 2, 25, 31, 55

[44] R. S. Sutton, "Temporal credit assignment in reinforcement learning," PhD thesis, University of Massachusetts Amherst, 1984. 1, 2, 9

[45] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, 2000. 8

[46] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. MIT Press, 2018. 5, 31

[47] I. Szita and A. Lörincz, "Learning tetris using the noisy cross-entropy method," *Neural Computation*, 2006. 19, 20

[48] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012. 27

[49] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical Review*, 1930. 29

[50] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," in *International Conference on Machine Learning*, 2017. 2, 55

[51] C. Watkins, "Learning from delayed rewards," PhD thesis, University of Cambridge, 1989. 6

[52] P. Wawrzyński, "Control policy with autocorrelated noise in reinforcement learning for robotics," *International Journal of Machine Learning and Computing*, 2015. 29

[53] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 1992. 1, 8

[54] Y. Wu, E. Mansimov, S. Liao, R. B. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," *arXiv preprint arXiv:1708.05144*, 2017. 55

Appendix A

Experiment Best Hyperparameters

In this section we report the best hyperparameter settings for the evaluated environments. We swept over policy (Actor) learning rate, action-value (Expert/Critic) learning rate, and other algorithm specific parameters.

Algorithm	Actor LR	Expert/Critic LR	Others
Actor-Expert (together)	5e-3	5e-2	-
Actor-Expert (separate)	1e-3	1e-1	-
Actor-Expert+ (together)	5e-3	1e-1	-
QT-Opt	-	1e-1	-
Actor-Expert (with PICNN)	1e-3	1e-2	-
PICNN	-	5e-2	-
NAF	-	1e-2	exploration scale (0.1)
Wire-fitting	-	1e-1	control points (100)
Optimal Q	-	1e-1	discretization (1e-3)
Soft Q-learning	5e-5	1e-1	reward scale (1000)
Actor-Critic (sample action)	5e-4	5e-2	-
Actor-Critic (mean action)	1e-3	5e-2	-
DDPG	5e-3	1e-1	-
Soft Actor-Critic (sample action)	1e-2	1e-1	reward scale (10)
Soft Actor-Critic (mean action)	1e-3	5e-1	reward scale (100)

Table A.1: Bimodal Environment

Algorithm	Actor LR	Expert/Critic LR	Others
Actor-Expert (together)	5e-4	1e-2	-
Actor-Expert (separate)	5e-4	5e-3	-
Actor-Expert+ (together)	5e-4	5e-3	-
Actor-Expert+ (separate)	1e-3	1e-2	-
QT-Opt	-	5e-3	-
PICNN	-	5e-3	-
NAF	-	1e-3	exploration scale (1.0)
Soft Q-learning	5e-3	1e-1	reward scale (100)
Actor-Critic (sample action)	1e-3	1e-2	-
Actor-Critic (mean action)	5e-4	1e-2	-
DDPG	5e-4	1e-2	-
Soft Actor-Critic (sample action)	1e-2	1e-2	reward scale (10)
Soft Actor-Critic (mean action)	5e-5	1e-2	reward scale (10)

Table A.2: Pendulum

Algorithm	Actor LR	Expert/Critic LR	Others
Actor-Expert (together)	1e-4	1e-3	-
Actor-Expert (separate)	1e-4	1e-3	-
Actor-Expert+ (together)	1e-4	1e-3	-
Actor-Expert+ (separate)	1e-4	1e-3	-
QT-Opt	-	1e-3	-
PICNN	-	1e-3	-
NAF	-	1e-3	exploration scale (0.3)
Soft Q-learning	1e-4	1e-3	reward scale (10)
Actor-Critic (sample action)	1e-4	1e-3	-
Actor-Critic (mean action)	1e-4	1e-3	-
DDPG	1e-4	1e-3	-
Soft Actor-Critic (sample action)	1e-4	1e-3	reward scale (10)
Soft Actor-Critic (mean action)	1e-4	1e-3	reward scale (10)

Table A.3: HalfCheetah

Algorithm	Actor LR	Expert/Critic LR	Others
Actor-Expert (together)	1e-5	1e-3	-
Actor-Expert (separate)	1e-4	1e-3	-
Actor-Expert+ (together)	1e-5	1e-3	-
Actor-Expert+ (separate)	1e-5	1e-3	-
QT-Opt	-	1e-3	-
PICNN	-	1e-4	-
NAF	-	1e-3	exploration scale (1.0)
Soft Q-learning	1e-4	1e-4	reward scale (1)
Actor-Critic (sample action)	1e-4	1e-3	-
Actor-Critic (mean action)	1e-5	1e-3	-
DDPG	1e-5	1e-3	-
Soft Actor-Critic (sample action)	1e-4	1e-3	reward scale (10)
Soft Actor-Critic (mean action)	1e-5	1e-3	reward scale (10)

Table A.4: Hopper

Algorithm	Actor LR	Expert/Critic LR	Others
Actor-Expert (together)	1e-5	1e-3	-
Actor-Expert (separate)	1e-5	1e-3	-
Actor-Expert+ (together)	1e-4	1e-4	-
Actor-Expert+ (separate)	1e-5	1e-3	-
QT-Opt	-	1e-3	-
PICNN	-	1e-4	-
NAF	-	1e-3	exploration scale (0.1)
Soft Q-learning	1e-3	1e-4	reward scale (100)
Actor-Critic (sample action)	1e-4	1e-3	-
Actor-Critic (mean action)	1e-5	1e-3	-
DDPG	1e-5	1e-3	-
Soft Actor-Critic (sample action)	1e-5	1e-4	reward scale (10)
Soft Actor-Critic (mean action)	1e-5	1e-3	reward scale (10)

Table A.5: Ant