Q-Functionals for Efficient Value-Based Continuous Control

Sreehari Rammohan
Department of Computer Science
Brown University
Providence, RI 02906
`sreehari@brown.edu`

Bowen He
Department of Computer Science
Brown University
Providence, RI 02906
`bowen_he@brown.edu`

Shangqun Yu
Department of Computer Science
Brown University
Providence, RI 02906
`shangqun_yu@brown.edu`

Sam Lobel
Department of Computer Science
Brown University
Providence, RI 02906
`sanuel_lobel@brown.edu`

George Konidaris
Department of Computer Science
Brown University
Providence, RI 02906
`gdk@cs.brown.edu`

Abstract

We present an alternative architecture for continuous control deep reinforcement learning which we call *Q-functionals*: instead of returning a single value for a state and action, our network transforms a state into a *function* that can be rapidly evaluated in parallel for *many* actions, allowing us to efficiently choose high-value actions through sampling alone. This contrasts with the typical architecture of off-policy reinforcement learning, where a policy network is trained for the sole purpose of selecting actions from the Q-function. We represent our action-dependent Q-function as a weighted sum of basis functions (Fourier, Polynomial, etc) over the action space, where the weights are state-dependent and output by the Q-functional network. In addition to fast sampling, this representation of state-action value is helpful in imposing an inductive bias on learning (such as making the value function smooth over actions) which can lead to faster speed of convergence. We characterize our framework, describe two implementations of Q-functionals, and demonstrate promising performance on a suite of continuous control tasks.

Keywords: Reinforcement Learning, Value-Based Continuous Control, Q-Functionals

1 Introduction

The final product of a successful reinforcement learning system is a *policy* that performs well at interacting with the environment, as measured by expected cumulative discounted reward. A dominant paradigm in reinforcement learning is to derive policies from a *Q-function* [8], which summarizes the expected cumulative reward for taking a given action. When the number of actions available is finite, a strong policy can be derived by enumerating all action-values for a given state and choosing the one with the highest value. However, this brute enumeration is impossible when there are large or infinite possible actions, for example when actions are drawn from a continuous vector space. This is the natural description of, for example, robotic locomotion and control; as such, a significant amount of effort has gone into action-selection in these domains. A common framework for so-called *continuous control* problems is to train a *policy network* that selects actions according to some criteria of the Q-values.

The training signal from a policy network comes completely from the Q-function, and as such they essentially summarize information about the Q-function for a given state so as to efficiently produce high-value actions. For example, standard practice is to train a policy network to estimate the *maximum*-valued action for any given state[5]. The quantities policy networks estimate are difficult to calculate on a per-state basis, so policy networks can be thought of as *amortizing* the expensive computation of finding desirable actions: the training procedure is expensive, but it allows actions to be *sampled* with a single pass of a neural network. We highlight two potential pitfalls of this framework. First, since the policy network is not reinitialized after every Q-function update, at any given timestep most of the policy training has been done on *old* versions of the Q-function; as such, the policy may be maximizing a *stale* estimate of action-value. Second, policies are generally either deterministic or sampled from a tight distribution around some single valuable action. Therefore, it is unlikely for the policy to sample two high-value actions if they are too far apart from each other [7]. This limits exploration as well as the robustness of the value-function across all actions.

We take a different perspective on this problem: instead of training a network to amortize expensive action-value computations, we design a learning system such that these evaluations are cheap to do in parallel. This opens up a new avenue for action-selection: instead of using a policy network, we can evaluate many actions in parallel and choose between them. Since the policy is directly derived from Q-values at each timestep it always reflects the most current action-value estimates. Furthermore, random sampling can easily generate high-value actions from throughout the action-space.

In this paper, we introduce a class of Q-functions we call *Q-functionals* that allow for efficient sampling. A *functional* is a function that returns another function. Likewise, a Q-functional transforms a state into a function over actions, such that Q-values for many actions can be evaluated in parallel and with little overhead. We describe two implementations of this architecture, and demonstrate its speed and effectiveness at continuous control reinforcement learning.

2 Q-Functionals

In RL, we seek to learn a *policy* which results in high cumulative discounted reward. For a given policy π, we define the state-action value function, or Q-function, as:

$$Q^{\pi}(s_t, a_t) = E_{\pi}[R(s_t, a_t) + \gamma V^{\pi}(s_{t+1})].$$

where $V^{\pi}(s)$ is the expected value of following π from state s. A common method of iteratively improving a Q function parameterized by θ is through *bootstrapping* [6]:

$$Q_{\theta}(s, a) \leftarrow r(s, a) + \gamma V^{\pi}(s')$$

Q-functionals are a way of expressing Q-functions so that many action-values can be evaluated in parallel for a given state. Consider a traditional Q-function, perhaps represented by a neural network:

$$Q(s, a) : (\mathcal{S} \times \mathcal{A}) \rightarrow \mathcal{R}.$$

A standard design for such a Q-function in continuous control is to concatenate states and actions, and then pass this through a 2-hidden-layer neural network. For this standard architecture, evaluating two action-values for the same state takes twice as long as evaluating one. A Q-functional breaks the computation of action-values into two parts: first, a state is transformed into parameters that define a function over the action space. Then, this function is evaluated for the action(s) in question:

$$Q_{\text{FUNC}}(s, a) : \mathcal{S} \rightarrow (\mathcal{A} \rightarrow \mathcal{R}) \tag{1}$$

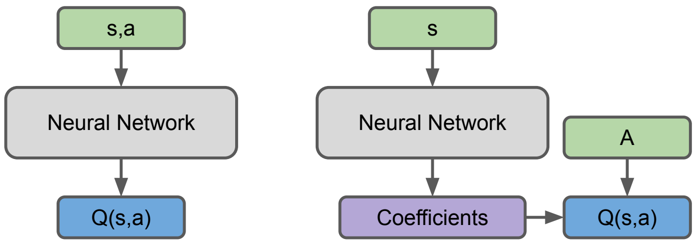

Figure 1: Network architecture of traditional Q-function (left) and Q-functional (right)

We design these functions such that while the per-state computation may be expensive, the per-action computation is relatively cheap. One simple way to represent these functions is by learning the coefficients of basis functions over the action space. An appropriate choice of basis function also adds an *inductive bias* to learning action-values: though we may expect values to have complicated dependence on state (especially when the state is something like an image), in general we expect values to be relatively smooth with respect to the action dimensions. In Figure 1, we visualize the difference between the two architectures as differing places that the actions enter the computation.

2.1 Fourier and Polynomial Basis Q-Functionals

We describe two implementations of Q-functionals using different basis functions. First, a Q-functional calculated using the Fourier basis is shown below in equation 2. The Fourier basis representation of *states* had great success when used for Q-learning in small state spaces [4]; we rely on the same inductive bias over the moderately-sized *action* spaces found in continuous control. L represents the difference between the max and min action space value allowed, $x_i(s), y_i(s)$ are the state-dependent coefficients output by the critic network. **C** is a matrix of frequencies for the Fourier basis. For a rank R Q-functional using the Fourier basis and operating in an environment with action dimension d, the full set of frequencies is represented by the cartesian product $\{0, 1, 2, ..., R\}^d$. In practice however, we find that limiting the set of frequencies to those where the sum of all elements in the frequency is less than or equal to the rank provided sufficient critic generalization. Formally speaking, $\mathbf{C} = \left\{ C_i = \{0, 1, 2, ..., R\}^d | \sum_{j=0}^{d} C_{ij} \leq R \right\}$

$$Q_{\text{FUNC}}(s, a) = \sum_{C_i \in \mathbf{C}} x_i(s) \sin(\frac{\pi}{L} a \cdot C_i) + y_i(s) \cos(\frac{\pi}{L} a \cdot C_i). \tag{2}$$

The learned coefficients $x_i(s)$ and $y_i(s)$ are dependent solely on the state, allowing us to efficiently sample many actions at once without needing to pass this through the coefficient critic network again. We can define a polynomial basis similarly: we represent our action-value function such that the total polynomial-order is less than some R:

$$Q(s, a) = \sum_{C_i \in \mathbf{C}} x_i(s) \prod_{j=0}^{d} a_j^{C_{ij}}. \tag{3}$$

Previous work demonstrates that action-values can be analytically maximized for quadratic polynomials over actions ($R = 2$) [2]. Without a policy network, however, Q-functions defined as such are restricted to convex functions over actions which can be overly limiting. Our sampling framework can derive a strong policy from more complex, higher-order polynomial Q-functions. In addition, as described in the next section, sampling allows us to calculate state-values with various methods that work better than simple maximization.

2.2 Extracting policies and state-values from Q-Functionals

In standard continuous-control RL, the policy network is used in two distinct locations: calculating state-value functions (bootstrapping), and during action-selection (policy-evaluation). Here we describe how a variety of strategies for both can be implemented using sampling.

The most common strategy for both is *action maximization*: aiming to find the highest-value action for every state, and using this for both bootstrapping and action-selection:

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s, a) \quad ; \quad V(s) = Q(s, \pi(s))$$

Instead of training a policy network, we can directly estimate the maximum through sampling k actions:

$$\pi(s) = \arg\max_{a_i \in \mathcal{A}^k} Q(s, a_i) \quad ; \quad V(s) = \max_{a_i \in \mathcal{A}^k} Q(s, a_i)$$

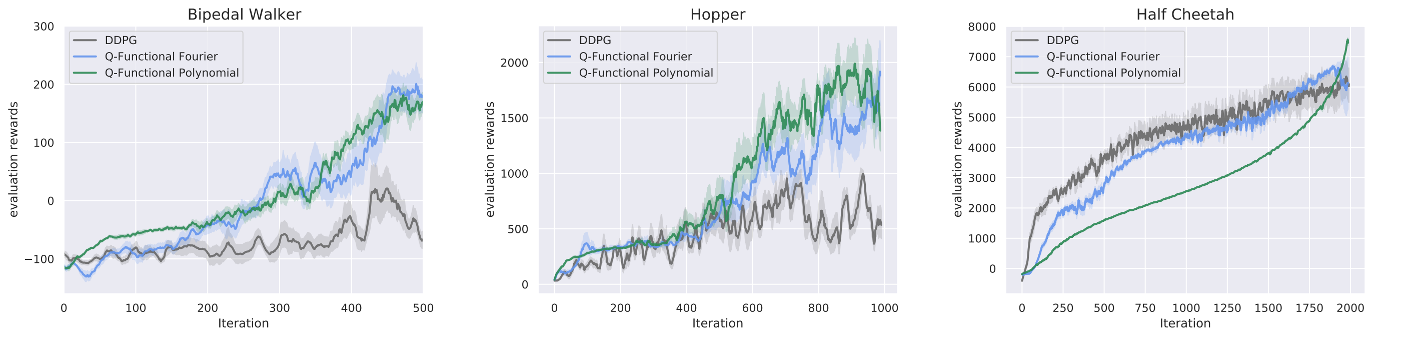

Figure 2: Q-functional Performance in comparison to DDPG. We perform 200 update steps every iteration with each iteration consisting of $\{1600, 1600, 2000\}$ interaction steps for Bipedal Walker, Hopper, and Half Cheetah respectively. Q-functionals all have "rank" 3. Results are averaged over 8 seeds, with the standard error shaded.

with accuracy bounds of $V(s)$ dependent on the dimension of $\mathcal{A}$, the Lipschitz-constant L of $Q(s,a)$, and k. For τ-entropy-regularized RL, the optimal policy is proportional to the Boltzmann distribution over Q-values. This simplifies to:

$$\pi(s) = \Pr(a|s) \leftarrow \frac{\exp\{Q(s,a)/\tau\}}{\int_{\mathcal{A}} \exp\{Q(s,a)/\tau\}} \quad ; \quad V(s) = \int_{\mathcal{A}} \pi(s)Q(s,a) - \log(\Pr(a|s)) = \tau \log\left(\int_{\mathcal{A}} \exp\{Q(s,a)/\tau\}\right)$$

Due to limitations imposed by common representations of stochastic policies (often constrained to state-dependent normal distributions over actions), this relation between policy and value cannot be achieved by standard policy-parameterizations [3]. In contrast, we can consistently estimate these qualities through Monte Carlo sampling:

$$\pi(s) = \Pr(a|s) = \frac{\exp\{Q(s,a)/\tau\}}{\frac{1}{k}\sum_{a_i \in \mathcal{A}^k} \exp\{Q(s,a)/\tau\}} \quad ; \quad V(s) = \tau \log\left(\frac{1}{k}\sum_{a_i \in \mathcal{A}^k} \exp\{Q(s,a)/\tau\}\right)$$

with accuracy bounds on both $\pi(s)$ and $V(s)$ computable from $Q_{\max} - Q_{\min}$, k, and τ.

In our experiments, we derive a robust policy $\pi(s)$ and value function $V(s)$ through a "top-n of k" approach: we evaluate k random actions, and represent our policy and value by sampling from the best n actions during interaction with the environment and averaging these n values during bootstrapping updates for stability.

$$\pi(s) \sim \text{top-n}\Big\{Q(s,a_1), ...Q(s,a_k)\Big\} \quad ; \quad V(s) = \text{mean}\Big[\text{top-n}\Big\{Q(s,a_1), ...Q(s,a_k)\Big\}\Big] \tag{4}$$

This is a similar strategy to TD3 [1], which computes $V(s)$ as a single-sample mean of the Q-values surrounding the policy-networks output. We find that averaging over *many* values, instead of simply choosing one, leads to decreased noisiness of $V(s)$ for the Bellman target in the update, and superior performance at maximizing reward.

3 Experimental Results

Our experimental results are included in Figure 2. All experiments are averaged over 8 seeds, with shading representing standard error confidence intervals. Both Fourier and Polynomial Q-functionals outperform Deep Deterministic Policy Gradients (DDPG) [5] on Bipedal Walker (4 dim action space), Hopper (3 dim action space), and Half Cheetah (6 dim action space). For action-selection and bootstrapping, we sample $k = 1,000$ actions for each state (which takes roughly equivalent computation to a single policy + value evaluation for DDPG). For $V(s)$ and our interaction policy $\pi(s)$ we anneal our top-n sampling in the range $[\frac{k}{2}, 1]$ over the course of training. We find this improves exploration when using Q-functionals; early in training the agent can try a broader range of actions. During evaluation, we choose the highest-value action ($n = 1$) sampled at each state.

As described earlier, standard feedforward Q-functions can be evaluated for multiple actions by simply evaluating the network multiple times. We quantify the speedup of the Q-functional paradigm compared to a standard feedforward Q function in Figure 2. A rank-3 Fourier critic is roughly 8 times faster than a standard neural network at this task on an Nvidia 2080ti GPU. Furthermore, the lightweight action-evaluation of Q-functionals allows for much larger sampling size: the standard architecture runs out of GPU memory with 5,000 samples, while the Fourier critic can evaluate up to 10,000 actions in a single pass. In the low-sample regime (up to 100), Q-functionals evaluate in near-constant time because the entire computation can be processed in parallel on a consumer GPU.

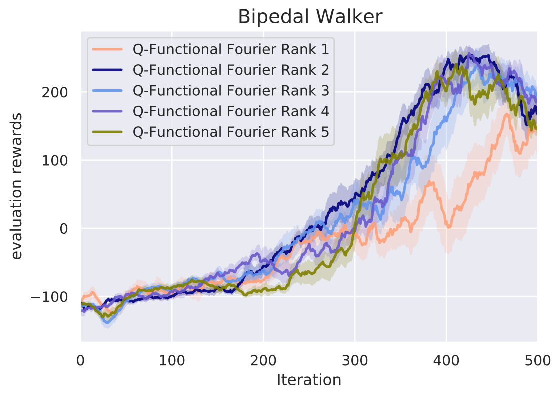

Figure 3: Rank Sweep for Bipedal Walker using Fourier Basis

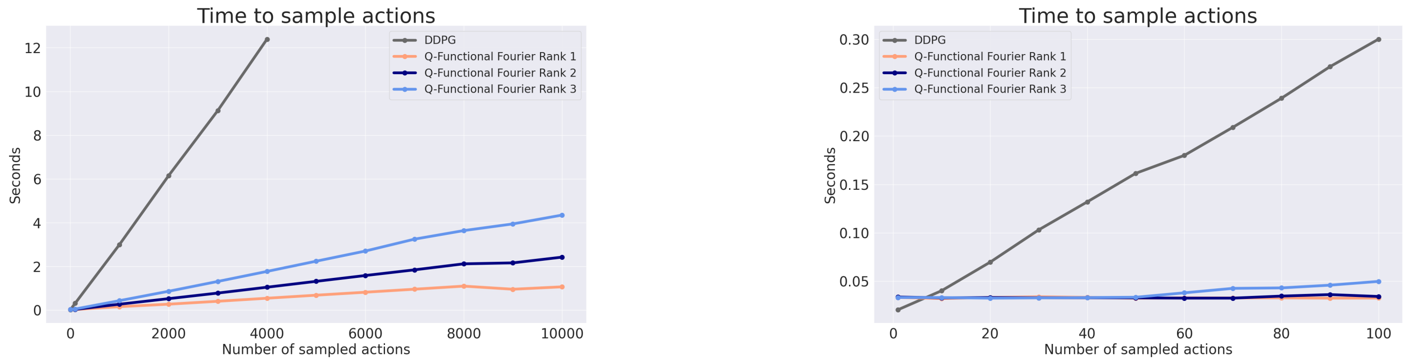

Figure 4: Speed comparison of 100 sets of action-evaluations of the same states (batch size 1024), for varying number of action-samples. **Left:** samples up to 10,000. **Right:** zoomed in sampling up to 100.

4 Discussion

We describe the framework of *Q-functionals*, a class of architectures for continuous-control RL that allow for efficiently calculating actions through *sampling*, without the need for training a large policy network. Besides requiring roughly half the parameters, our method naturally leads to more thorough exploration, represents the most *current* estimates of the Q-value, and enforces a smooth inductive bias over the action-space. We find that implementations of this framework consistently outperform baseline policy gradient methods on a range of continuous control tasks.

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv e-prints*, page arXiv:1802.09477, February 2018.

[2] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv e-prints*, page arXiv:1603.00748, March 2016.

[3] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[4] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv e-prints*, page arXiv:1509.02971, September 2015.

[6] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[7] Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.