Deep Radial-Basis Value Functions for Continuous Control

Kavosh Asadi[1,2], **Neev Parikh**[2], **Ronald E. Parr**[3], **George D. Konidaris**[2], **Michael L. Littman**[2]

[1] Amazon Web Services
[2] Department of Computer Science, Brown University
[3] Department of Computer Science, Duke University

Abstract

A core operation in reinforcement learning (RL) is finding an action that is optimal with respect to a learned value function. This operation is often challenging when the learned value function takes continuous actions as input. We introduce deep radial-basis value functions (RBVFs): value functions learned using a deep network with a radial-basis function (RBF) output layer. We show that the maximum action-value with respect to a deep RBVF can be approximated easily and accurately. Moreover, deep RBVFs can represent any true value function owing to their support for universal function approximation. We extend the standard DQN algorithm to continuous control by endowing the agent with a deep RBVF. We show that the resultant agent, called RBF-DQN, significantly outperforms value-function-only baselines, and is competitive with state-of-the-art actor-critic algorithms.

Introduction

A fundamental object of interest in RL is the value (Q) function, which quantifies the expected return for taking action a in state s. Many RL algorithms, such as Q-learning (Watkins 1989) and actor-critic (Barto, Sutton, and Anderson 1983), learn an approximation of the Q function from environmental interactions. When using function approximation to learn the Q function, the agent has a parameterized function class, and learning consists of finding a parameter setting θ for the approximate value function $\widehat{Q}(s, a; \theta)$ that accurately represents the true Q function. A core operation here is finding an optimal action with respect to the learned value function, specifically $\arg\max_{a \in \mathcal{A}} \widehat{Q}(s, a; \theta)$. The need for performing this operation arises not just for action selection, but also when learning $\widehat{Q}$ itself via bootstrapping (Sutton and Barto 2018).

The optimization problem $\arg\max_{a \in \mathcal{A}} \widehat{Q}(s, a; \theta)$ is generally challenging if the action space $\mathcal{A}$ is continuous, because the learned value function $\widehat{Q}(s, a; \theta)$ could have many local maxima and saddle points; therefore, naïve approaches such as gradient ascent can be expensive and inaccurate (Lim et al. 2018). Value-function-only algorithms have thus remained under-explored in continuous control.

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

It is trivial to solve $\arg\max_{a \in \mathcal{A}} \widehat{Q}(s, a; \theta)$ when $\mathcal{A}$ is discrete. Therefore, a simple approach for continuous actions is to partition the continuous action space. By performing this discretization, we can essentially treat the original continuous problem as a discrete one. While this approach can be effective in highly constrained settings, to uniformly cover the action space, we need a partition size that grows exponentially with the number of action dimensions. Pazis and Parr (2011) attempt to combat this issue via approximate linear programming, but scaling this approach to large domains remains open.

Another line of work has shown the benefits of using function classes that are conducive to efficient action maximization. For example, Gu et al. (2016) explored function classes that can capture an arbitrary dependence on the state, but only a quadratic dependence on the action. Given a quadratic action dependence, Gu et al. (2016) showed how to compute $\arg\max_{a \in \mathcal{A}} \widehat{Q}(s, a; \theta)$ quickly. A more general idea is to use input–convex neural networks (Amos, Xu, and Kolter 2017) that restrict $\widehat{Q}(s, a; \theta)$ to convex (or concave) functions with respect to a, so that the maximization problem can be solved efficiently using convex-optimization techniques (Boyd and Vandenberghe 2004). Unfortunately, these approaches are unable to support universal function approximation (Hornik, Stinchcombe, and White 1989), and may lead into inaccurate value functions regardless of the amount of experience provided to the agent.

We introduce deep radial-basis value functions (RBVFs): Q functions approximated by a standard deep neural network augmented with an RBF output layer. We show that deep RBVFs enable us to efficiently and accurately identify an approximately optimal action without impeding universal function approximation.

Prior work in RL used RBVFs for problems with continuous states. (See, for example, Geramifard et al. (2013) or Section 9.5.5 of Sutton and Barto (2018) for a discussion.) These results often use RBFs to transform the original state space to a higher-dimensional space in which linear function approximation yields accurate results. Our use of RBFs is fundamentally different in two ways: a) we apply RBFs to the action space not the state space, and b) we use deep neural networks to learn the centroids that constitute the RBFs. Applying RBFs to the action space allows us to

show that the global maximum of $\widehat{Q}(s, \cdot; \theta)$ is approximately equal to the value of the best centroid—$\max_a \widehat{Q}(s, a; \theta) \approx \max_{i \in [1,N]} \widehat{Q}(s, a_i; \theta)$ where a_i represents the ith centroid. Because computing $\max_{i \in [1,N]} \widehat{Q}(s, a_i; \theta)$ is easy in an RBVF, this result provides tremendous leverage.

The aim of our experimental results are twofold. We first endow DQN, a standard deep RL algorithm originally proposed for discrete actions (Mnih et al. 2015), with a deep RBVF and produce a new continuous-control algorithm called RBF-DQN. We evaluate RBF-DQN to demonstrate its superior performance relative to value-function-only baselines, and to show its competitiveness with state-of-the-art actor-critic deep RL. We also show that a deep RBVF could serve as the critic in standard actor-critic algorithms such as DDPG (Silver et al. 2014; Lillicrap et al. 2016). This enables us to approximate the greedy value function via the critic, and ultimately improve the performance of DDPG.

Background

We study the interaction between an environment and an agent that seeks to maximize reward (Sutton and Barto 2018), a problem typically formulated using Markov Decision Processes (MDPs) (Puterman 2014). An MDP is specified by a tuple: $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$. In this work, $\mathcal{S}$ and $\mathcal{A}$ denote the continuous state space and the continuous action space of the MDP, and we further assume that they are closed and bounded, therefore compact. The goal of an RL agent is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that collects high sums of discounted rewards across timesteps.

For a state $s \in \mathcal{S}$, action $a \in \mathcal{A}$, and a policy π, we define the state–action value function:

$$Q^\pi(s, a) := \mathbf{E}_\pi \big[G_t \mid s_t = s, a_t = a \big] \ ,$$

where $G_t := \sum_{i=t}^{\infty} \gamma^{i-t} R_i$ is called the *return* at timestep t. The state–action value function of an *optimal* policy, denoted by $Q^*(s, a)$, can be written recursively (Bellman 1952):

$$Q^*(s, a) = R(s, a) + \gamma \int_{s'} T(s, a, s') \max_{a'} Q^*(s', a') ds' .$$

In the absence of a model, a class of RL algorithms solve for the fixed point of Bellman's equation using environmental interactions. Q-learning (Watkins 1989), a notable example of these so-called model-free algorithms, learns an approximation of Q^*, denoted by $\widehat{Q}$ and parameterized by θ. Under function approximation, θ parameters are updated as follows:

$$\begin{aligned} &\theta \leftarrow \theta + \alpha \, \delta \, \nabla_\theta \widehat{Q}(s, a; \theta) \ , \\ &\text{where} \quad \delta := r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s', a'; \theta) - \widehat{Q}(s, a; \theta) \ , \end{aligned} \tag{1}$$

using tuples of experience $\langle s, a, r, s' \rangle$ collected during environmental interactions. Note that Q-learning's update rule (1) is agnostic to the choice of function class, and so in principle any differentiable and parameterized function class could be used in conjunction with the above update to learn θ parameters. For example, Sutton (1996) used linear function approximation, Konidaris, Osentoski, and Thomas (2011) used Fourier basis functions, and Mnih et al. (2015) used convolutional neural networks. Finally, we assume that $\widehat{Q}$ is continuous.

Deep Radial-Basis Value Functions

Deep Radial-Basis Value Functions (RBVFs) combine the practical advantages of deep networks (Goodfellow, Bengio, and Courville 2016) with the theoretical advantages of radial-basis functions (RBFs) (Powell 1987). A deep RBVF is comprised of a number of arbitrary hidden layers, followed by an ***RBF output layer***, defined next. The RBF output layer, first introduced in a seminal paper by Broomhead and Lowe (1988), is sometimes used as a standalone single-layer function approximator, referred to as a (shallow) RBF network. It is also a core ingredient of the kernel trick in Support Vector Machines (Cortes and Vapnik 1995). We use an RBF network as the final, or output, layer of a deep network.

For a given input a, the RBF layer $f(a)$ is defined as:

$$f(a) := \sum_{i=1}^{N} g(a - a_i) \, v_i \ , \tag{2}$$

where each a_i represents a *centroid* location, v_i is the value of the centroid a_i, N is the number of centroids, and g is an RBF. A commonly used RBF is the negative exponential:

$$g(a - a_i) := e^{-\beta \|a - a_i\|} \ , \tag{3}$$

equipped with an inverse smoothing parameter $\beta \geq 0$. (See Karayiannis (1999) for a thorough treatment of other RBFs.) Formulation (2) could be thought of as an interpolation based on the value and the weights of all centroids, where the weight of each centroid is determined by its proximity to the input. Proximity here is quantified by the RBF g, in this case the negative exponential (3).

It is theoretically useful to normalize centroid weights to ensure that they sum to 1 so that f implements a weighted average. This weighted average is sometimes referred to as a normalized RBF layer (Moody and Darken 1989; Bugmann 1998):

$$f_\beta(a) := \frac{\sum_{i=1}^{N} e^{-\beta \|a - a_i\|} \, v_i}{\sum_{i=1}^{N} e^{-\beta \|a - a_i\|}} \ . \tag{4}$$

As the inverse smoothing parameter $\beta \to \infty$, the function implements a winner-take-all case where the value of the function at a given input is determined only by the value of the closest centroid location, nearest-neighbor style. This limiting case is sometimes referred to as a *Voronoi decomposition* (Aurenhammer 1991). Conversely, f converges to the mean of centroid values regardless of the input a as β gets close to 0; that is, $\forall a \ \lim_{\beta \to 0} f_\beta(a) = \frac{\sum_{i=1}^{N} v_i}{N}$. Since an RBF layer is differentiable, it could be used in conjunction with gradient-based optimization techniques and Backprop to learn the centroid locations and their values by optimizing for a loss function. Note that formulation (4) is different than the Boltzmann softmax operator (Asadi and Littman 2017;

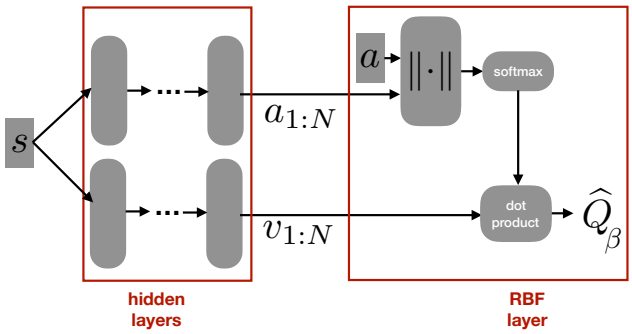


Figure 1: Architecture of a deep RBVF, which could be thought of as an RBF output layer added to an otherwise standard deep Q function. All operations of the final RBF layer are differentiable, so the parameters of hidden layers θ, which represent the mappings $a_i(s;\theta)$ and $v_i(s;\theta)$, can be learned using gradient-based optimization techniques.

Song, Parr, and Carin 2019), where the weights are determined not by an RBF, but by the action values.

Finally, to represent the Q function for RL, we use the following formulation:

$$\widehat{Q}_\beta(s,a;\theta) := \frac{\sum_{i=1}^{N} e^{-\beta\|a-a_i(s;\theta)\|}\, v_i(s;\theta)}{\sum_{i=1}^{N} e^{-\beta\|a-a_i(s;\theta)\|}} \,. \qquad (5)$$

From equation (5) a deep RBVF learns two mappings: state-dependent centroid locations $a_i(s;\theta)$ and state-dependent centroid values $v_i(s;\theta)$. The role of the output layer is to compute the output of the entire deep RBVF. We illustrate the architecture of a deep RBVF in Figure 1. In the experimental section, we demonstrate how to learn parameters θ.

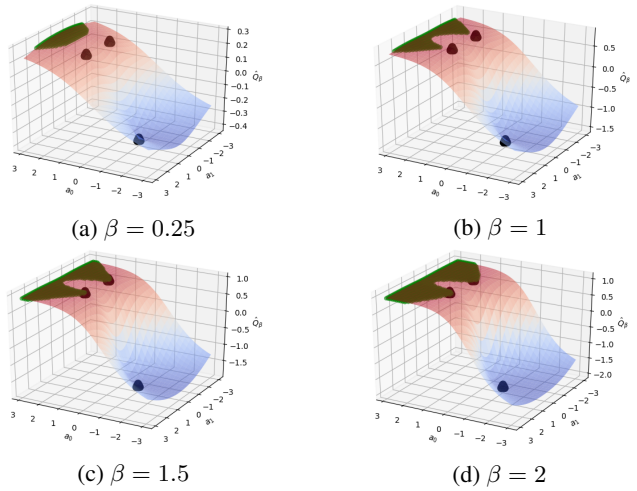
We now show that deep RBVFs have a highly desirable property for value-function-based RL, namely that they enable easy action maximization.

First, it is easy to find the output of a deep RBVF at each centroid location a_i, that is, to compute $\widehat{Q}_\beta(s,a_i;\theta)$. Note that $\widehat{Q}_\beta(s,a_i;\theta) \neq v_i(s;\theta)$ in general for a finite β, because the other centroids $a_j\ \forall j \in \{1,..,N\}-i$ may have non-zero weights at a_i. To compute $\widehat{Q}_\beta(s,a_i;\theta)$, we access the centroid location using $a_i(s;\theta)$, then input a_i to get $\widehat{Q}(s,a_i;\theta)$. Once we have $\widehat{Q}(s,a_i;\theta)\ \forall a_i$, we can trivially find the best centroid : $\max_{i\in[1,N]} \widehat{Q}_\beta(s,a_i;\theta)$.

Recall that our goal is to compute $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta)$, but so far we have shown how to compute $\max_{i\in[1,N]} \widehat{Q}_\beta(s,a_i;\theta)$. We now show that these two quantities are equivalent in one-dimensional action spaces. More importantly, Theorem 1 shows that with arbitrary number of dimensions, there may be a gap, but that this gap gets exponentially small with increasing the inverse smoothing parameter β. Proofs are in the Appendix.

Theorem 1. *Let $\widehat{Q}_\beta$ be a normalized negative-exponential RBVF.*

1. *For $\mathcal{A} \in \mathcal{R}$:*
   $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta) = \max_{i\in[1,N]} \widehat{Q}_\beta(s,a_i;\theta)$.



(a) $\beta = 0.25$ (b) $\beta = 1$ (c) $\beta = 1.5$ (d) $\beta = 2$

Figure 2: The output of a RBVF with 3 fixed centroid locations and values, but different settings of the inverse smoothing parameter β on a 2-dimensional action space. The regions in dark green highlight the set of actions a for which a is extremely close to the global maximum, $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta)$. Observe the reduction of the gap between $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta)$ and $\max_{i\in[1,N]} \widehat{Q}_\beta(s,a_i;\theta)$ by increasing β, as guaranteed by Theorem 1.

2. *For $\mathcal{A} \in \mathcal{R}^d \quad \forall d > 1$:*
   $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta) - \max_{i\in[1,N]} \widehat{Q}_\beta(s,a_i;\theta) \leq \mathcal{O}(e^{-\beta})$.

In light of Theorem 1, to approximate $\max_{a\in\mathcal{A}} \widehat{Q}(s,a;\theta)$ we simply compute $\max_{i\in[1,N]} \widehat{Q}(s,a_i;\theta)$. The accuracy of the approximation hinges on the magnitude of the inverse smoothing parameter. Notice that using extremely high values of β may lead to poor generalization error due to the complexity of the resultant function class. Therefore, the theorem should be used as a guarantee on the approximation error, not as a justification for using extremely large β values. Notice also that this result holds for normalized negative-exponential RBVFs, but not necessarily for the unnormalized case or for RBFs other than negative exponential.

In Theorem 2, presented in the Appendix, we show that RBVFs support universal function approximation. Collectively, Theorems 1 and 2 guarantee that deep RBVFs ensure accurate and efficient action maximization without impeding universal function approximation. This combination of properties stands in contrast with prior work that used function classes that enable easy action maximization but lack UFA (Gu et al. 2016; Amos, Xu, and Kolter 2017), as well as prior work that preserved the UFA property but did not guarantee arbitrarily high accuracy when performing the maximization step (Lim et al. 2018; Ryu et al. 2020).

In terms of scalability, note that the RBVF formulation scales naturally owing to its freedom to determine centroids that best minimize the loss function. As a thought experiment, suppose that some region of the action space has a high value, so an agent with greedy action selection fre-

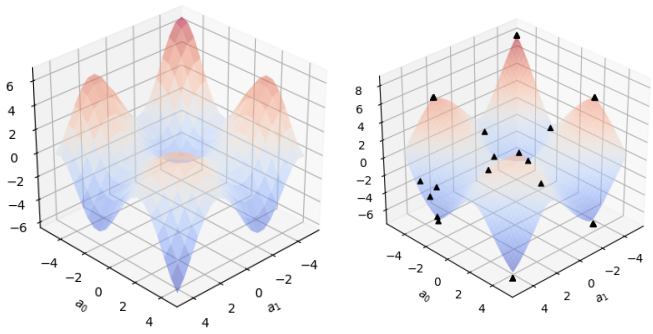


Figure 3: Left: True surface of the function. Right: The accurate approximation learned by the RBVF. Black dots represent the centroids. Note also that $\max_{i\in[1,N]} \widehat{r}_\beta(a_i;\theta)$ is an accurate approximation for $\max_{a\in\mathcal{A}} r(a;\theta)$.

quently chooses actions from that region. A deep RBVF would then move more centroids to the region, because the region heavily contributes to the loss function. Since the centroids are state-dependent, the network can learn to move the centroids to more rewarding regions on a per-state basis. It is unnecessary to initialize centorid locations carefully, or to uniformly cover the action space *a priori*. In this sense, learning RBVFs could be thought of as a form of adaptive and soft discretization learned by gradient descent. Adaptive discretization techniques have proven fruitful in terms of sample-complexity guarantees in bandit problems (Kleinberg, Slivkins, and Upfal 2008), as well as in RL (Sinclair, Banerjee, and Yu 2019; Touati, Taiga, and Bellemare 2020; Domingues et al. 2020).

Experiments

In this section, we empirically assess the effectiveness of RBVFs in the context of non-linear regression, value-function-only deep RL, and actor-critic deep RL.

RBVFs for Regression

To demonstrate the operation of an RBF network in the clearest setting, we start with a regression task where the agent is provided with sample input–output pairs $\langle a, r\rangle$. We use the function: $r(a) = \|a\|_2 \frac{\sin(a_0)+\sin(a_1)}{2}$, whose surface is presented in Figure 3 (left). It is non-convex and includes several local maxima (and minima). We sampled 500 inputs a uniformly randomly from $[-3, 3]^2$ to obtain a dataset for training. We used the mean-squared-error loss, and used $N = 20$. The surface is learned well (Figure 3).

RBVFs for Value-Function-Only Deep RL

We now use deep RBVFs for solving continuous-action RL problems. To this end, we learn a deep RBVF using a learning algorithm akin to that of DQN (Mnih et al. 2015), but extended to the continuous-action case. DQN uses the following loss function for learning the value function:

$$L(\theta) := \mathbf{E}_{s,a,r,s'}\left[\left(r+\gamma \max_{a'\in\mathcal{A}} \widehat{Q}(s',a';\theta^-) - \widehat{Q}(s,a;\theta)\right)^2\right].$$

DQN adds tuples of experience $\langle s, a, r, s'\rangle$ to a buffer, and later samples a minibatch of tuples to compute $\nabla_\theta L(\theta)$. DQN maintains a second network parameterized by weights θ^-. This second network, denoted $\widehat{Q}(\cdot,\cdot,\theta^-)$ and referred to as the *target network*, is periodically synchronized with the online network $\widehat{Q}(\cdot,\cdot,\theta)$. RBF-DQN uses the same loss function, but modifies the function class of DQN. Concretely, DQN learns a deep network with one output per action, exploiting the discrete and finite nature of the action space. By contrast, RBF-DQN takes a state and an action vector, and outputs a single scalar using a deep RBVF. The pseudo-code for RBF-DQN is presented in Algorithm 1.

Algorithm 1 Pseudo-code for RBF-DQN

```
Initialize deep RBVF with N, β, θ
Initialize replay buffer B, ϵ, γ, α, α⁻, θ⁻
for E episodes do
    Initialize s
    while not done do
        a ← ϵ-greedy(Q̂_β(s, ·; θ), ϵ)
        s', r, done ← env.step(s, a)
        add ⟨s, a, r, s', done⟩ to B; s ← s'
    end while
    for M minibatches sampled from B do
        for ⟨s, a, r, s', done⟩ in minibatch do
            Δ = (r − Q̂_β(s, a; θ)) ∇_θ Q̂(s, a; θ)
            if not done then
                get centroids a_i(s'; θ⁻), i ∈ [1, N]
                Δ += γ max_i Q̂_β(s', a_i(s'; θ⁻); θ⁻)
            end if
            θ ← θ + αΔ · ∇_θ Q̂_β(s, a; θ)
        end for
        θ⁻ ← (1 − α⁻)θ⁻ + α⁻θ
    end for
end for
```

We now evaluate RBF-DQN against state-of-the-art value-function-only deep RL baselines. We understand NAF (Gu et al. 2016) and ICNN (Amos, Xu, and Kolter 2017) as two of the best continuous-action extensions of DQN in the RL literature, so we used them as our baselines. For RBF-DQN, as well as each of the two baselines, we performed 1000 updates per episodes when each episode ends. The authors of ICNN have released an official code base [1], which we used to obtain the ICNN learning curves. To the best of our knowledge, Gu et al. (2016) did not release code, but we have used a public implementation of NAF [2]. Compared to the reported results in the ICNN paper, we were able to roughly achieve the same performance for NAF and ICNN. However, for completeness, we show, via dashed horizontal lines, levels of performance for NAF and ICNN reported by the ICNN paper whenever one was present. Another relevant baseline is CAQL (Ryu et al. 2020), for which we could neither find the authors' code nor a public implementation. Since the original CAQL paper used environ-

[1] github.com/locuslab/icnn

[2] github.com/ikostrikov/pytorch-ddpg-naf

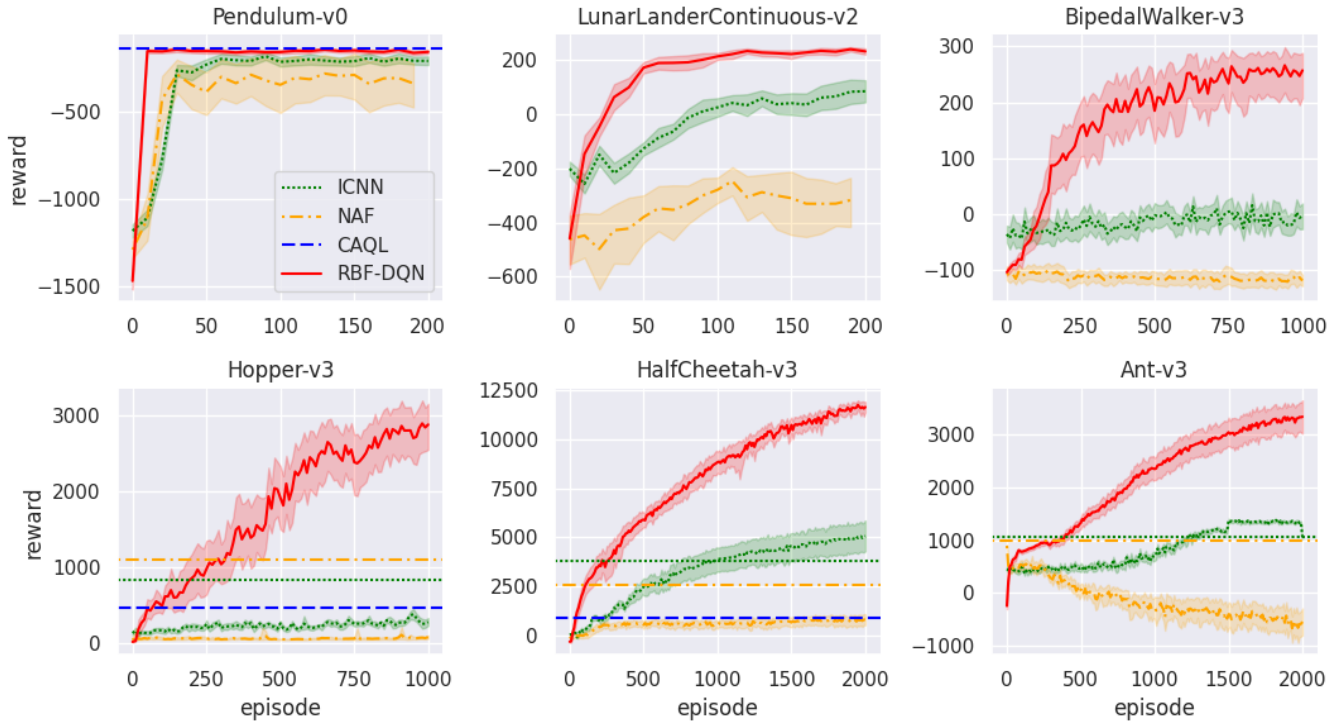


Figure 4: A comparison between RBF-DQN and value-function-only deep RL baselines on 6 Open AI Gym environments. For ICNN and NAF, we have included two results: Dashed lines indicate the level of performance reported by Amos, Xu, and Kolter (2017), and the learning curves show the performance of the baselines obtained by running publicly-available implementations. All runs for this figure and all following figures are averaged over 20 trials with different random seeds.

ments with constrained action spaces, we only compare with the best reported results for Hopper-v3, HalfCheetah-v3, and Pendulum-v0, as these environments were used with the full action space. We have released our code: [3]. Details of our hyper-parameter tuning can be found in the Appendix. RBF-DQN is clearly outperforming the two baselines, even when considering the results reported by the ICNN authors (Amos, Xu, and Kolter 2017).

In light of the above results, we can claim that RBF-DQN is demonstrably a state-of-the-art value-function-only deep RL algorithm, but how does RBF-DQN compare to state-of-the-art actor-critic deep RL? To answer this question, we compared RBF-DQN to state-of-the-art actor critic deep RL, namely DDPG (Silver et al. 2014; Lillicrap et al. 2016), TD3 (Fujimoto, Hoof, and Meger 2018), and SAC (Haarnoja et al. 2018). For TD3 and DDPG, we used the official code released by Fujimoto, Hoof, and Meger (2018)[4], which is also used by numerous other papers. For SAC, we used a public implementation.[5] In Figure 5, we show, not just the learning curves we obtained by running the implementations, but also results reported in the TD3 and SAC papers.

We note that the authors of TD3, DDPG, and SAC report their results in steps. In our experiments, we report results in episodes and have modified all implementations to have 1000 gradient updates per episode, while continuing to record steps. We plot horizontal lines at the level of reward obtained by the authors (and reported in their papers) at the average steps reached by the publicly available implementations when run for the specified number of episodes. For example, in Ant-v3, SAC reaches 1,076,890 steps on average when run for 2000 episodes. Therefore, we estimate the reward from the authors' graph at around 1.07×10^6 steps to be roughly 4000. More details for all relevant domains are available in the Appendix.

From Figure 5, RBF-DQN is performing better than or is competitive with state-of-the-art actor critic deep RL baselines. RBF-DQN can compete with these baselines despite the fact that it only uses 2 neural networks (an online value function and a target value function), while SAC and TD3, for instance, use 5 and 6 networks, respectively. In our experiments, TD3 and SAC perform 1000 gradient updates on two critics and one actor per each episode, while RBF-DQN only updates a deep RBVF 1000 times. Therefore, TD3 and SAC are also performing 3 times as many updates as RBF-DQN. Lastly, some of the ideas leveraged by TD3 and SAC, such as value clipping in TD3 and SAC, and entropy regularization in SAC, can be integrated into RBF-DQN. We believe these combinations are promising, and leave the investigation of these combinations for future work.

[3] github.com/kavosh8/RBFDQN_pytorch
[4] github.com/sfujim/TD3
[5] github.com/pranz24/pytorch-soft-actor-critic

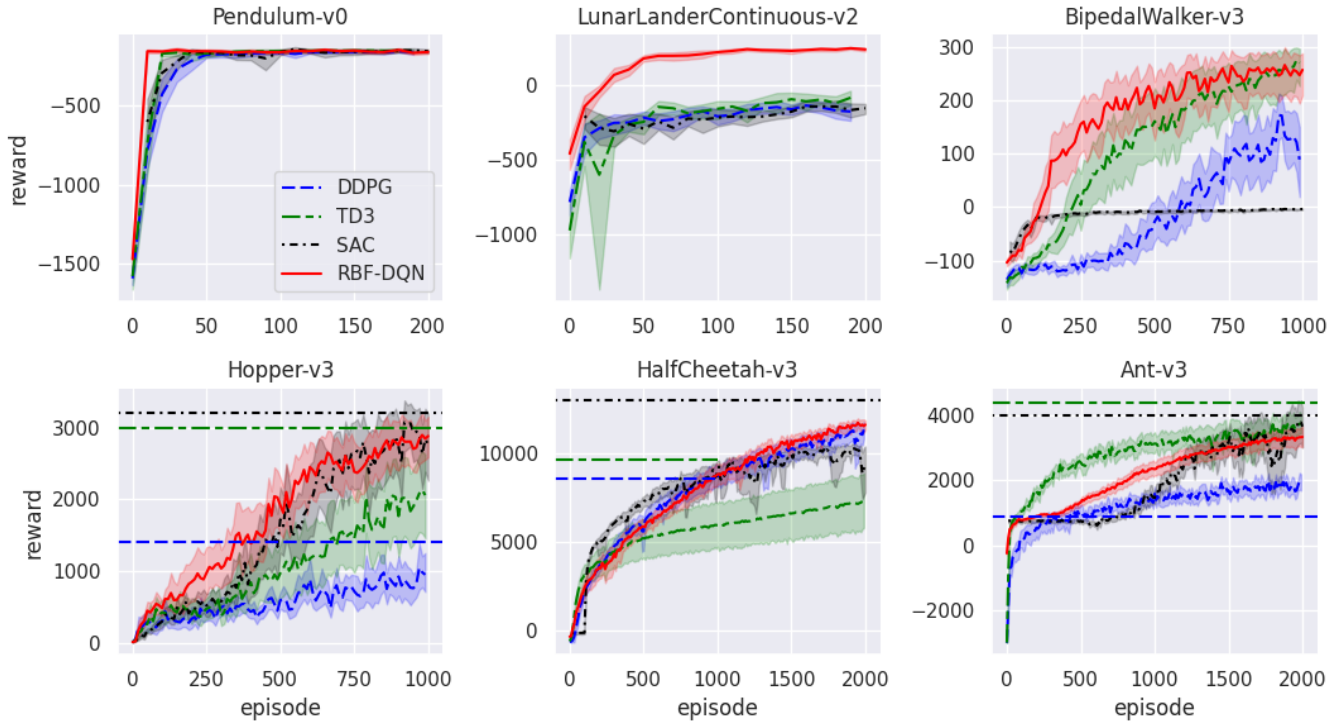


Figure 5: A comparison between RBF-DQN and state-of-the-art actor-critic deep RL.

Investigating the effects of β and N

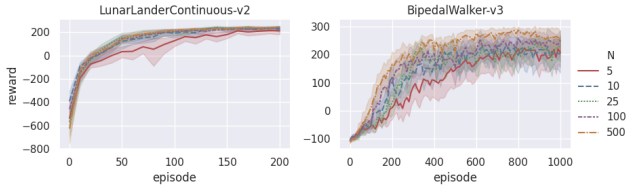


Figure 6: RBF-DQN performance as a function of the number of centroids N.

Figure 7: RBF-DQN performance (area under curve) as a function of the number of centroids N.

We introduced two new hyper-parameters for RBVFs, namely the number of centroids N, and the inverse smoothing parameter β. To better understand the impact of these hyper-parameters, we investigate the performance of the agent as a function of these hyper-parameters. We first show

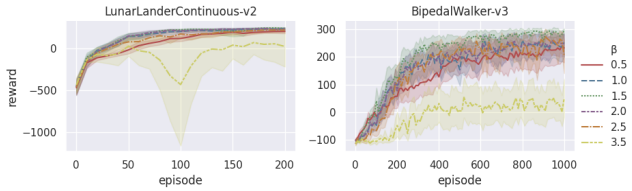


Figure 8: RBF-DQN performance as a function of the inverse smoothing parameter β.

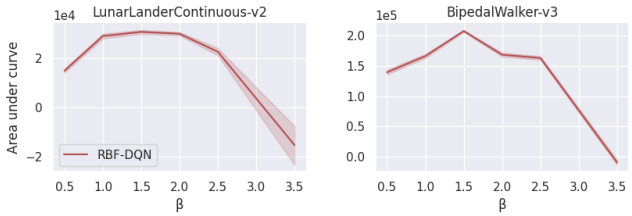


Figure 9: RBF-DQN performance (area under curve) as a function of the inverse smoothing parameter β.

the performance of RBF-DQN as a function of N (Figure 6 and Figure 7). We note two interesting observations. First, the agent can perform reasonably well even with a very small number of centroids such as $N = 5$. Second, we see improved performance as we increase the number of centroids. The first observation suggests that we can get good results even when massive computation is not available and

we have to use very small N. The second property is also appealing, because, in the presence of enough compute power, we can improve the agent's performance by simply increasing N.

In contrast, with respect to β, the inverse smoothing parameter of the negative exponential, we often see an inverted-U shape, with the best setting at some intermediate value (see Figure 8 and Figure 9). We note that, while using a large value of β makes it theoretically possible to approximate any function up to any desired accuracy, the trade-off between using small β values that make unwarranted generalization, and large β values that yield extremely local approximations make intermediate values of β work best, as demonstrated.

RBVFs for Actor-Critic Deep RL

So far, we have applied RBVFs to value-function-only RL, but can RBVFs be useful for other RL algorithms? To answer this question affirmatively, we now use RBVFs for actor-critic deep RL, in particular as the critic in the DDPG algorithm (Silver et al. 2014; Lillicrap et al. 2016). Recall that, in contrast with value-function-only RL, actor-critic algorithms learn two separate networks: a value function (or the critic), and a policy (or the actor) that is mainly used for action selection. Silver et al. (2014) introduces the deterministic policy-gradient actor critic, but makes the observation that, in continuous control, computing the greedy action with respect to the learned critic is not tractable (see their Subsection 3.1), leading them to instead use a SARSA update for the critic. We showed that a deep RBVF approximately solves this maximization problem tractably. Leveraging this insight, we can modify the DDPG algorithm to use an RBVF critic. Given a tuple $\langle s, a, r, s' \rangle$, we can learn, via the critic, the value function with a Q-learning or SARSA update:

$$\begin{aligned} \theta &\leftarrow \theta + \alpha\, \delta_{\text{RBF-DDPG}}\, \nabla_\theta \widehat{Q}(s, a; \theta)\ , \\ \delta_{\text{Q-learning}} &:= r + \gamma \max_i \widehat{Q}(s', a_i(s'); \theta) - \widehat{Q}(s, a; \theta) \\ \delta_{\text{SARSA}} &:= r + \gamma \widehat{Q}(s', \pi(s'; \omega); \theta) - \widehat{Q}(s, a; \theta)\ . \end{aligned} \tag{6}$$

The original DDPG algorithm used the SARSA update, obviating the action-maximization step. Using a deep RBVF, RBF-DDPG performs the Q-learning update shown above to update the critic. RBF-DDPG is otherwise analogous to DDPG.

We compare RBF-DDPG and DDPG in Figure 10. It is clear that the use of RBVFs benefits DDPG. Although preliminary evaluations of RBF-DDPG do not exceed state-of-the-art performance, further investigation is required to see if the addition of other algorithmic ideas, such as those presented in Fujimoto, Hoof, and Meger (2018) and Haarnoja et al. (2018), can further improve the performance of RBF-DDPG.

Conclusion

We introduced deep radial-basis value functions (RBVFs), and showed that they enable us to extend the Q-learning update from discrete-action settings to continuous control. Leaning on this insight, we introduced two new RL algorithms, RBF-DQN and RBF-DDPG. We showed that, in particular, RBF-DQN is significantly better than value-function-only deep RL baselines. RBF-DQN is also competitive with state-of-the-art actor-critic deep RL despite maintaining fewer networks and performing fewer updates. RBVFs facilitate action maximization, do not impede universal function approximation, and scale to large action spaces. Deep RBVFs are thus an appealing choice for value function approximation in continuous control.

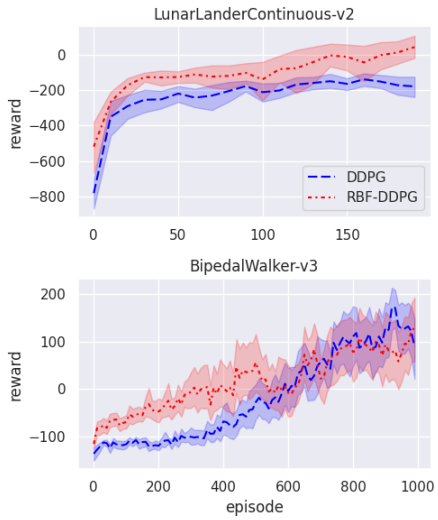


Figure 10: A comparison between DDPG and RBF-DDPG.

Future Work

We envision several promising directions for future work. First, the RL literature has numerous examples of algorithmic ideas that help improve value-function-only algorithms (see Hessel et al. (2018) for some examples). These ideas are usually proposed for domains with discrete actions, so extending them to continuous-action domains using RBVFs could be an exciting direction to pursue. Our results set the stage to exploring these extensions.

We solely focused on deep RBVFs with negative exponentials, but various RBFs exist in the literature (Karayiannis 1999). We also envision future work where we learn an abstract action representation, akin to Chandak et al. (2019), and utilize RBFs that operate in the abstract action space. This can better leverage the underlying structure of the action space.

Different exploration strategies exist for value-function-only RL: optimistic initialization (Sutton and Barto 2018; Machado, Srinivasan, and Bowling 2015), softmax policies (Rummery and Niranjan 1994; Sutton and Barto 2018), uncertainty-based exploration (Osband et al. 2016), and Zooming (Kleinberg, Slivkins, and Upfal 2008). A combination of advanced exploration strategies with deep RBVFs could prove more effective than dithering strategies used in this paper.

Acknowledgements

We are appreciative of the feedback provided by many of our colleagues. In particular, we thank Dilip Arumugam for his comments in the early steps of the work, as well as Sam Lobel and Seungchan Kim for our invaluable discussions.

References

Amos, B.; Xu, L.; and Kolter, J. Z. 2017. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, 146–155.

Asadi, K.; and Littman, M. L. 2017. An alternative softmax operator for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, 243–252.

Aurenhammer, F. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys* 23(3): 345–405.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics* 834–846.

Bellman, R. 1952. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America* 38(8): 716.

Boyd, S.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge University Press.

Broomhead, D. S.; and Lowe, D. 1988. Radial Basis Functions, multi-variable functional interpolation and adaptive networks. *Complex Systems* .

Bugmann, G. 1998. Normalized Gaussian Radial Basis Function networks. *Neurocomputing* 20(1-3): 97–110.

Chandak, Y.; Theocharous, G.; Kostas, J.; Jordan, S. M.; and Thomas, P. S. 2019. Learning action representations for reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, 941–950.

Cortes, C.; and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3): 273–297.

Domingues, O. D.; Ménard, P.; Pirotta, M.; Kaufmann, E.; and Valko, M. 2020. Regret bounds for kernel-based reinforcement learning. *arXiv preprint arXiv:2004.05599* .

Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing function approximation error in Actor-Critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, 1587–1596.

Geramifard, A.; Walsh, T. J.; Tellex, S.; Chowdhary, G.; Roy, N.; and How, J. P. 2013. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning* 6(4): 375–451.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Gu, S.; Lillicrap, T.; Sutskever, I.; and Levine, S. 2016. Continuous deep Q-learning with model-based acceleration. In *Proceedings of the 33rd International Conference on Machine Learning*, 2829–2838.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 1856–1865.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5): 359–366.

Karayiannis, N. B. 1999. Reformulated radial basis neural networks trained by gradient descent. *IEEE Transactions on Neural Networks* 10(3): 657–671.

Kleinberg, R.; Slivkins, A.; and Upfal, E. 2008. Multi-armed bandits in metric spaces. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, 681–690.

Konidaris, G.; Osentoski, S.; and Thomas, P. 2011. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations*.

Lim, S.; Joseph, A.; Le, L.; Pan, Y.; and White, M. 2018. Actor-Expert: A framework for using cction-value methods in continuous action spaces. *arXiv preprint arXiv:1810.09103* .

Machado, M. C.; Srinivasan, S.; and Bowling, M. 2015. Domain-independent optimistic initialization for reinforcement learning. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

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

Moody, J.; and Darken, C. J. 1989. Fast learning in networks of locally-tuned processing units. *Neural computation* 1(2): 281–294.

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep exploration via bootstrapped DQN. In *Advances in neural information processing systems*, 4026–4034.

Pazis, J.; and Parr, R. 2011. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning*, 1185–1192.

Powell, M. J. 1987. Radial Basis Functions for multivariable interpolation: A review. *Algorithms for Approximation* .

Puterman, M. L. 2014. *Markov Decision Processes.: Discrete stochastic dynamic programming*. John Wiley & Sons.

Rummery, G. A.; and Niranjan, M. 1994. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England.

Ryu, M.; Chow, Y.; Anderson, R.; Tjandraatmadja, C.; and Boutilier, C. 2020. CAQL: Continuous Action Q-Learning. In *Proceedings of the 8th International Conference on Learning Representations*.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, 387–395.

Sinclair, S. R.; Banerjee, S.; and Yu, C. L. 2019. Adaptive Discretization for Episodic Reinforcement Learning in Metric Spaces. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1–44.

Song, Z.; Parr, R.; and Carin, L. 2019. Revisiting the softmax Bellman operator: new benefits and new perspectives. In *Proceedings of the 36th International Conference on Machine Learning*, 5916–5925.

Sutton, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, 1038–1044.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Touati, A.; Taiga, A. A.; and Bellemare, M. G. 2020. Zooming for Efficient Model-Free Reinforcement Learning in Metric Spaces. *arXiv preprint arXiv:2003.04069* .

Watkins, C. 1989. Learning from Delayed Rewards. *King's College, Cambridge* .