Rainbow RBF-DQN

Sreehari Rammohan [*1] **Bowen He** [*1] **Shangqun Yu** [*2] **Eric Hsiung** [*2] **Eric Rosen** [1] **George Konidaris** [1]

Abstract

Deep reinforcement learning has been extensively studied, resulting in several extensions to DQN that improve its performance, such as replay buffer sampling strategies, distributional value representations, and double/dueling networks. Previous works have examined these extensions in the context of either discrete action spaces or in conjunction with actor-critic learning algorithms, but there has been no investigation of combining them for deep value-based continuous control. We adapted the methods discussed in Rainbow DQN to RBF-DQN, a deep value-based method for continuous control, showing improvements when evaluated over a set of 7 OpenAI Gym continuous control tasks in baseline performance and sample efficiency. Rainbow RBF-DQN outperforms both vanilla RBF-DQN and state-of-the-art actor-critic methods on the most challenging tasks such as Half-Cheetah and Humanoid.

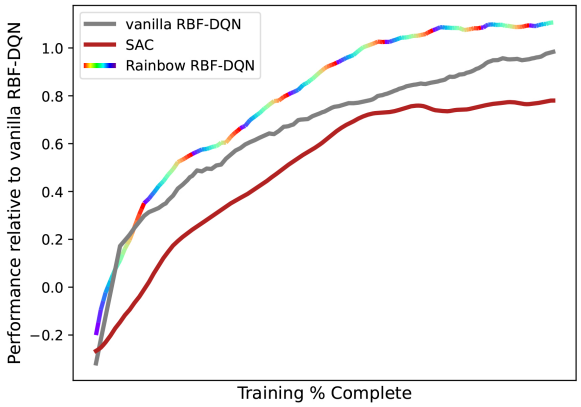

Figure 1: Rainbow RBF-DQN and SAC performance relative to vanilla RBF-DQN across the 7 MuJoCo control tasks presented.

1. Introduction

Deep Reinforcement Learning (DRL) algorithms have sucessfully learned to solve complex decision-making problems, in part due to algorithmic extensions to the core Q-learning framework. For example, Mnih et al. (2013) demonstrated that by using a target network in conjunction with a replay buffer to save previous transitions, a Convolutional Neural Network (CNN) could be trained using Q-Learning (i.e: DQN) to effectively play Atari games from raw pixels. Other important algorithmic extensions are replay buffer sampling techniques (Schaul et al., 2015), double networks (van Hasselt et al., 2015), and distributional representations for value functions (Badia et al., 2020). Rainbow DQN (Hessel et al., 2018) selected six popular DQN techniques—Double Q-Learning, Prioritized Replay (Schaul et al., 2015), Dueling Networks (Wang et al., 2015), Multi-Step Learning (Asis et al., 2018), Distributional Reinforcement Learning (Bellemare et al., 2017), and Noisy Nets (Fortunato et al., 2018)—and demonstrated that these techniques can be merged effectively into a single integrated agent that achieves high performance across 57 Atari games.

The introduction of RBF-DQN (Asadi et al., 2020) was a key neural network architectural breakthrough that made deep value based RL perform competitively on continuous action domains. In this paper we select four of the improvements to DQN—Distributional Representations, Dueling Networks, Double Networks, and Priority Experience Replay—and modify them to work in the continuous action space with RBF-DQN. The resulting agent, termed Rainbow RBF-DQN, overall demonstrates more stable learning and lower sample complexity on 7 OpenAI/Mujoco continuous control tasks: Humanoid, Ant, Half-Cheetah, Hopper, Bipedal Walker, Lunar Lander, and Pendulum.

2. Background and Related Work

2.1. Markov Decision Processes and Q-Learning

Sequential decision making problems are often modeled as Markov Decision Processes (MDPs)—represented as a tuple of $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ represents the set of possible states in the environment an agent could access, $\mathcal{A}$ represents the set of possible actions the agent can take, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ captures a stochastic notion of transition dynamics for an agent acting in a state, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ represents the scalar reward received from the environment due to a transition, and $\gamma \in [0, 1]$

represents the discounting of future reward. In reinforcement learning, the objective of an agent is to find a policy which maximizes its expected sum of discounted rewards, known as expected return. The state-value function $V^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s = s_t]$ represents the agent's expected return when starting in state $s \in \mathcal{S}$ and acting forever under policy π. The state-action-value function $Q^\pi(s,a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s = s_t, a = a_t]$ represents the agent's expected return when taking action $a \in \mathcal{A}$ from state s, and then forever acting under policy π. An optimal policy π^* is the policy for which $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in \mathcal{S}$. Value-based approaches find optimal policies by solving the Bellman equation for either the optimal $V(s)$ or optimal $Q(s,a)$. Q-Learning (Watkins & Dayan, 2004) solves $Q^*(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} \max_{a'} Q^*(s',a')$. For discrete action spaces, the max operation is tractable by iterating through finite $|\mathcal{A}|$ actions. In continuous action spaces however, function approximations must be employed.

Deep value-based methods use deep learning to learn a set of parameters θ to approximate the state-action value function $Q_\theta(s,a)$. The vanilla Q-Learning objective is to minimize the TD-Error (also known as the Bellman error), which is defined for a single transition (s,a,r,s') as $r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)$. Q-Learning is off-policy and therefore allows the agent to use a behavioral policy (such as epsilon greedy) to approximate a target policy (the optimal policy). Mnih et al. (2013) introduced DQN, allowing deep neural networks to play Atari using the TD-Error objective, a replay buffer to store previously experienced transitions, and a target network for stabilizing the TD target between learning updates. Note that in Q-Learning, the update rule relies on finding $\max_{a' \in \mathcal{A}} \hat{Q}(s',a';\theta)$. This is prohibitively expensive in continuous action spaces, due to an infinite search space, and approximations like discretizing the action space may produce sub-optimal solutions.

RBF-DQN (Asadi et al., 2020) is a value-based deep reinforcement learning approach that uses radial-basis functions to approximate the Q function for continuous action space tasks. RBF-DQN learns both the radial-basis centroids (which represent sampled actions) and their values, and uses a kernel to efficiently compute the action that maximizes the Q-value with a bounded error that depends on the temperature parameter β. Specifically, RBF-DQN approximates $Q^*(s,a)$ by learning centroid locations $a_i(s;\theta)$ and centroid values $v_i(s;\theta)$ as functions of state s and parameters θ and β according to:

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

The centroid locations $a_i(s;\theta)$ and state-dependent centroid values $v_i(s;\theta)$ are used to form the Q function output (Asadi et al., 2020), and are learned end-to-end during training by optimizing the Bellman error similar to the DQN loss (Mnih et al., 2013). The action maximization property of RBF-DQN (Asadi et al., 2020) guarantees all critical points of $\hat{Q}_\beta$ can be well-approximated by a centroid location a_i. This makes action-maximization as simple as searching over all N centroids $\max_{i \in [1,N]} \hat{Q}_\beta(s,a_i;\theta)$ where a_i represents a centroid location. In multi-dimensional action spaces, the temperature parameter β can be tuned to ensure an upper bound on error for the optimal action (Asadi et al., 2020):

$$\max_{a \in \mathcal{A}} \hat{Q}_\beta(s,a;\theta) - \max_{i \in [1,N]} \hat{Q}_\beta(s,a;\theta) \leq \mathcal{O}(e^{-\beta}). \quad (2)$$

The action-maximization property of Deep RBFs, paired with its universal function approximating properties, make RBF-DQN well-suited to continuous control tasks.

2.2. Deep Reinforcement Learning Improvements

Many of the recent successes in deep reinforcement learning have come from improvements to the vanilla Q-Learning approach. In this work, we specifically investigate a set of popular extensions to DQN that are included in Rainbow DQN (Hessel et al., 2018), which combines double Q-Learning (van Hasselt et al., 2015), prioritized experience replay (Schaul et al., 2015), dueling networks (Wang et al., 2015), and distributional RL (Bellemare et al., 2017). We now describe each of these Rainbow extensions before describing how we augment them to be effective for working with RBF-DQN.

Double Q-Learning: Under certain stochastic environments, Q-Learning can perform very poorly due to the overestimation of action values. Double Q-Learning networks use two estimators to address the overestimation bias. Double-DQN (van Hasselt, 2010) uses an online network, parameterized by θ, to pick the action that is evaluated by the target network, parameterized by θ^-. This leads to an adapted DQN loss:

$$(r_{t+1} + \gamma Q_{\theta^-}(s_{t+1}, \operatorname*{argmax}_a Q_\theta(s_{t+1},a)) - Q_\theta(s_t,a_t))^2. \quad (3)$$

Related work has shown that the double Q-Learning algorithm can be generalized to work with function approximators (van Hasselt et al., 2015).

Dueling Networks Dueling Networks (Wang et al., 2015) separate out the process of learning state-action pairs $Q(s,a)$ into learning state value $V(s)$ and action advantage values $A(s,a)$. Each stream of computation shares the same initial layers for featurizing the input (parameterized by θ), but they then split into two separate sequences of neural network layers that compute estimates of the state value (parameterized by β) and advantage values (parameterized by α) separately. The two streams are then combined to

produce a single Q function estimate using the following equation:

$$Q(s,a;\theta,\alpha,\beta) = V(s,\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a^*} A(s,a^*;\theta,\alpha)). \quad (4)$$

Note that by subtracting off the advantage of the best action, the issue of identifiability is resolved since the advantage function will have zero advantage at the chosen action (Wang et al., 2015). Since the output of a Dueling Network is a single Q-value, it can be integrated with any other algorithmic improvement for value-based DRL.

Prioritized Experience Replay In Priority Experience Replay (Schaul et al., 2015), samples are prioritized during training based on their respective TD-error. This means that samples which the agent has difficulty estimating the correct value of, are replayed more frequently. There are two important hyperparameters: α, to what extent priority is incorporated into the uniform sampling process, and β, the compensation factor for performing importance sampling on the TD-error for the transitions in the buffer.

Distributional RL with Quantile Regression Dabney et al. (2017) proposed learning an N quantile distribution over Q values. As opposed to the C51 algorithm from Bellemare et al. (2017) which learns the probabilities associated with 51 fixed supports (ranging from $v_{\min}$ to $v_{\max}$), this scheme fixes the probabilities associated with each support $\left(\frac{1}{N}\right)$ and instead learns the support values. Because the set of supports for the predicted and target quantiles can be disjoint, the Wasserstein loss is used (as opposed to the KL divergence in Bellemare et al. (2017)).

Due to space constraints, please see the appendix section A.2 for the related work on **Noisy Networks and Multi-Step Returns**.

3. Extending RBF-DQN

We investigated how to extend the improvements of Rainbow DQN to RBF-DQN. In this section, we present new augmentations to accommodate applying Double Q-Learning, Dueling Networks, and Distributional Q-Learning to RBF-DQN. All other Rainbow augmentations (PER, Noisy Networks, and Multi-step Learning) are directly applicable without modification. [1]

3.1. Double Q-Learning

To adapt double Q-Learning networks for RBF-DQN, the optimal action is selected using the online centroid and centroid value modules, parameterized by θ, and then producing a Q-value using the centroid and centroid values from the target network, parameterized by θ^-, respectively. We implement double RBF-DQN according to

$$Q(s', a^*; \theta^-) = \frac{\sum_{i=1}^{N} e^{-\beta||a^* - a_i(s';\theta^-)||} v_i(s';\theta^-)}{\sum_{i=1}^{N} e^{-\beta||a^* - a_i(s';\theta^-)||}} \quad (5)$$

where $a^* = \text{argmax}_{a_i} Q(s', a_i; \theta)$ is chosen according to the online network. The action a^* is the centroid location associated with the highest Q value according to Equation 1.

3.2. Distributional Q-Learning

In distributional Q-learning, rather than learning Q values in expectation, we learn the complete value distribution for a given state-action pair. We considered two approaches for learning this value distribution:

1. Fix the supports of the distribution and learn the probabilities associated with each support. This was the approach used in the C51 algorithm (Bellemare et al., 2017), where 51 supports are uniformly arranged between $v_{\min}$ and $v_{\max}$. We found that agents using this approach plateaued in performance, and got stuck in local optima. The min and max support values for the distribution are important, as a bad choice of these hyper-parameters makes it extremely difficult or impossible to represent certain values (since the value is taken to be the mean of this distribution). Furthermore, during training the value distribution will be varying wildly from state to state as the network is learning so there is no one $v_{\min}$ and $v_{\max}$ that is suitable across training.

2. Fix the probabilities of each support (uniformly across all supports) and learn the values correspondingly. This approach is more flexible than the one above, with fewer hyper-parameters to tune.

Applying this to RBF-DQN, the network's centroid value module is modified to instead output the quantile distribution for each centroid, denoted by the matrix $\mathbf{Z} \in \mathbb{R}^{N_C \times N_Q}$ values, where N_C is the number of centroids and N_Q is the number of quantiles. Taking the expectation of the supports for the ith centroid results in the value v_i for the ith centroid, and can be summarized by $\mathbf{v}^\top = \mathbf{w}^\top \mathbf{Z}$, where $\mathbf{v} \in \mathbb{R}^{N_Q}$, a is the proposed action and a_i is the location of the ith centroid, and the components w_i of the vector $\mathbf{w} \in \mathbb{R}^{N_C}$ represent the normalized distances from the proposed action a to each centroid location a_i:

$$w_i = \left(\frac{\|a_i - a\|_2}{\sum_{i=1}^{N_C} \|a_i - a\|_2} \right) \quad (6)$$

[1] https://github.com/SreehariRamMohan/rainbow_RBFDQN. Our rainbow RBF-DQN implementation.

$$\begin{bmatrix} w_1 & w_2 & ... & w_{N_C} \end{bmatrix} \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1N_Q} \\ z_{21} & z_{22} & \cdots & z_{2N_Q} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N_C1} & z_{N_C2} & \cdots & z_{N_CN_Q} \end{bmatrix} \tag{7}$$

Each row in $\mathbf{Z}$ represents the quantiles for a given centroid. The element $z_{ij}(s, a; \theta')$ represents the jth quantile for the ith centroid. $\mathbf{v}$ represents the weighted quantile distribution for an action a, which can then be averaged to get $Q(s, a)$.

Quantile regression loss (Dabney et al., 2017) is used to update the network, where τ is a quantile probability associated with the given target distribution.

$$\rho_\tau(u) = |\tau - \delta_{u<0}|\mathcal{L}(u),$$

where

$$\mathcal{L}(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| < 1 \\ |u| - \frac{1}{2}, & \text{otherwise.} \end{cases}$$

In algorithm 1, u is the distance between each predicted support and each target support, with an additional argument of τ to specify which quantile point the prediction support is approximated to. Each quantile point $z_j(s, a; \theta')\ \forall j \in [1, N_Q]$ is independently moved toward the correct quantile position (according to the target quantile distribution formed by $\mathcal{T}z_j$ where $\mathcal{T}$ is the Bellman operator).

3.3. Integrated Agent

Our integrated RBF-DQN agent (Rainbow RBF-DQN) incorporates Distributional Quantile Regression, Double Networks, and Priority Experience Replay (PER). Empirically we found that Dueling networks, Noisy networks, and Multi-step returns did not lead to a consistent boost in performance over vanilla RBF-DQN so we exclude them from our Rainbow agent.

In order to incorporate PER to distributional RBF-DQN, the sampling weights are replaced with the regression loss for each batch of transitions.

$$p_t \propto (\sum_{i=1}^{N_Q} \mathbb{E}_j[\rho_{\tau_i}(r + \gamma z_j(s', a^*; \theta^-) - z_i((s, a; \theta))])^w.$$

Double Q learning is added to Distributional RBF-DQN by having the online network parameterized by θ choose the centroid with the highest value at s', and return that centroid as the optimal action. This action is then passed into the target network parameterized by θ^- to return the target quantile distribution. $y_{\text{target}} = r + \gamma z_j(s', a^*; \theta^-)$, where $a^* \leftarrow \text{argmax}\, \mathbb{E}[z_i(s', \cdot; \theta)]$.

Algorithm 1
Distributional RBF-DQN with Quantile Regression

Initialize deep RBVF with N_C, N_Q, β, θ
Initialize replay buffer $\mathcal{D}, \epsilon, \gamma, \theta^-$
Initialize $maxstep$, F_{update}, $F_{synchronize}$
$steps \leftarrow 0$
$\theta^- \leftarrow \theta$
while $steps < maxstep$ **do**
  $s \leftarrow$ env.reset(), $done \leftarrow False$
  while not done **do**
    $a \sim \epsilon$-greedy$\big(Q_\beta(s, \cdot; \theta)\big)$
    $s', r, done \leftarrow$ env.step(s, a)
    add $\langle s, a, r, s', done \rangle$ to $\mathcal{D}$
    if $steps\%F_{update} == 0$ **then**
      sample a batch from $\mathcal{D}$
      $L \leftarrow 0$
      for $\langle s, a, r, s', done \rangle$ in batch **do**
        $Q_\beta(s', a'; \theta^-) \coloneqq 1/N_Q \sum_j z_j(s', a'; \theta^-)$
        $a^* \leftarrow \text{argmax}_{a'}\ \mathrm{Q}_\beta(s', a'; \theta^-)$
        $\mathcal{T}z_j \leftarrow r + \gamma z_j(s', a^*; \theta^-), \forall j$
        $L+ = \sum_{i=1}^{N_Q} \mathbb{E}_j[\rho_{\tau_i}(\mathcal{T}z_j - z_i((s, a; \theta))]$
      end for
      minimize L using gradient decent
    end if
    if $steps\%F_{synchronize} == 0$ **then**
      $\theta^- \leftarrow \theta$
    end if
    $steps+ = 1$
  end while
end while

4. Experiments

4.1. Rainbow RBF-DQN for Regression

To illustrate the distributional component of Rainbow RBF-DQN, we perform a stochastic regression task modelled after a simple stateless bandit problem, with the discount factor $\gamma = 0$. The agent randomly selects an action $a = (x, y) \in [-2, 2]^2$ and receives a stochastic reward $r(x, y \mid U)$ conditioned on a uniform random variable $U \sim \mathcal{U}(0, 1)$. During training, we hold the state s to a constant 1 as the agent minimizes $\|r(x, y \mid U) - \hat{Q}_\theta(s, a)\|_2^2$. The agent must accurately predict the payoff based on the reward function described below, with components shown in Figure 2:

$$r(x, y \mid U = u) = \begin{cases} \sin(x^2 + y^2) & \text{if } u < \frac{1}{3} \\ 2 & \text{if } \frac{1}{3} \leq u \leq \frac{2}{3} \\ ([\frac{x}{10}]^{\frac{1}{3}} + 1.5) & \text{if } \frac{2}{3} < u \end{cases}$$

Training quantile distributional RBF-DQN on $r(x, y)$ gives an estimation shown in Figure 3. We evaluated the agent at the following points in the action space $\{(-1.8, 0), (0, 0), (1.8, 0)\}$ producing 3 distributions—

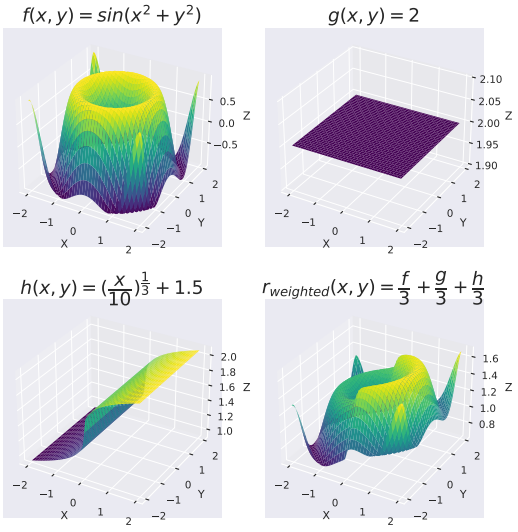

Figure 2: Reward function components used in the stochastic regression. Each function has a uniform random chance of being used as the reward function.

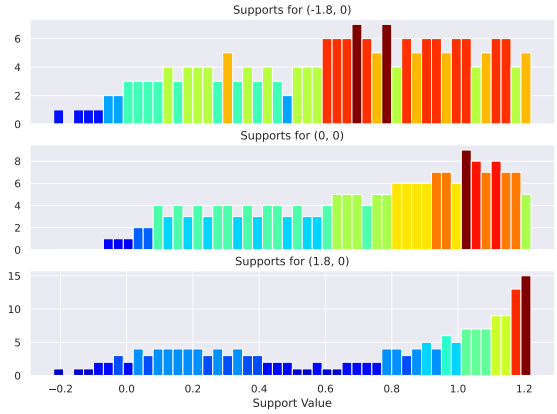

Figure 3: The histogram above shows support location frequency. Hotter colors represent clusters of supports around a particular value.

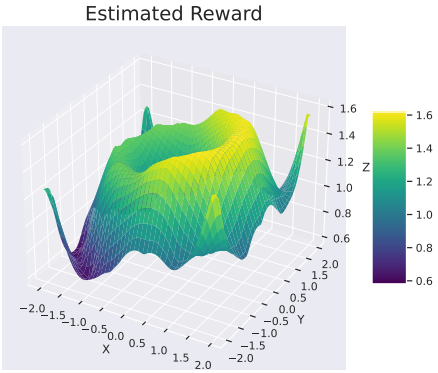

Figure 4: Distributional RBF-DQN trained with 200 centroids

higher frequency implies a gathering of quantiles at those locations.

At the point $(-1.8, 0)$, $\mathbb{E}[r(-1.8, 0)] = 0.997$, however based on the f, g, h reward components found in Figure 2 it is clear to see the reward distribution is trimodal, with payoffs of $\sin((-1.8)^2)$, 2, or $\left(\frac{-1.8}{10}\right)^{\frac{1}{3}} + 1.5$ depending on the value of u. Notice how the top support distribution in Figure 3 for the point $(-1.8, 0)$ also reflects the trimodal nature of the reward, while the expectation of the distribution is close to the true 0.997 (as can be seen in Figure 4).

4.2. MuJoCo Control Tasks

We test our rainbow RBF-DQN agent on the 6 Open AI gym (Brockman et al., 2016) tasks presented in the vanilla RBF-DQN paper (Asadi et al., 2020) as well as on Humanoid-v2, a challenging 17 dimensional action space control task (for hyper-parameter details see appendix section A.5). For each task we use the low dimensional state space and default dense reward. We report the cumulative evaluation rewards across 10 trajectories (evaluation frequency can be found in section A.5 Table 9).

For RBF-DQN with PER, performance was generally best with β annealed from 0.4 at the start of training to 1.0 by the end of training and α set to 0.1. The combined hyper-parameter space for Rainbow RBF-DQN is extremely large and we supplied best estimate values, it may be possible to squeeze out even more performance from these agents with a systematic tuning.

5. Analysis

In this section we review our experimental results and compare rainbow RBF-DQN to vanilla RBF-DQN and an existing state of the art actor-critic method, Soft Actor Critic (SAC) from Haarnoja et al. (2018). We use the public implementation of vanilla RBF-DQN[2] and use the stable baselines 3 implementation of SAC[3].

5.1. Comparison to baseline vanilla RBF-DQN and SAC

In Figure 1 we plot rainbow RBF-DQN's performance across all 7 control tasks relative to vanilla RBF-DQN and

[2]https://github.com/kavosh8/RBFDQN_pytorch
[3]https://github.com/DLR-RM/stable-baselines3

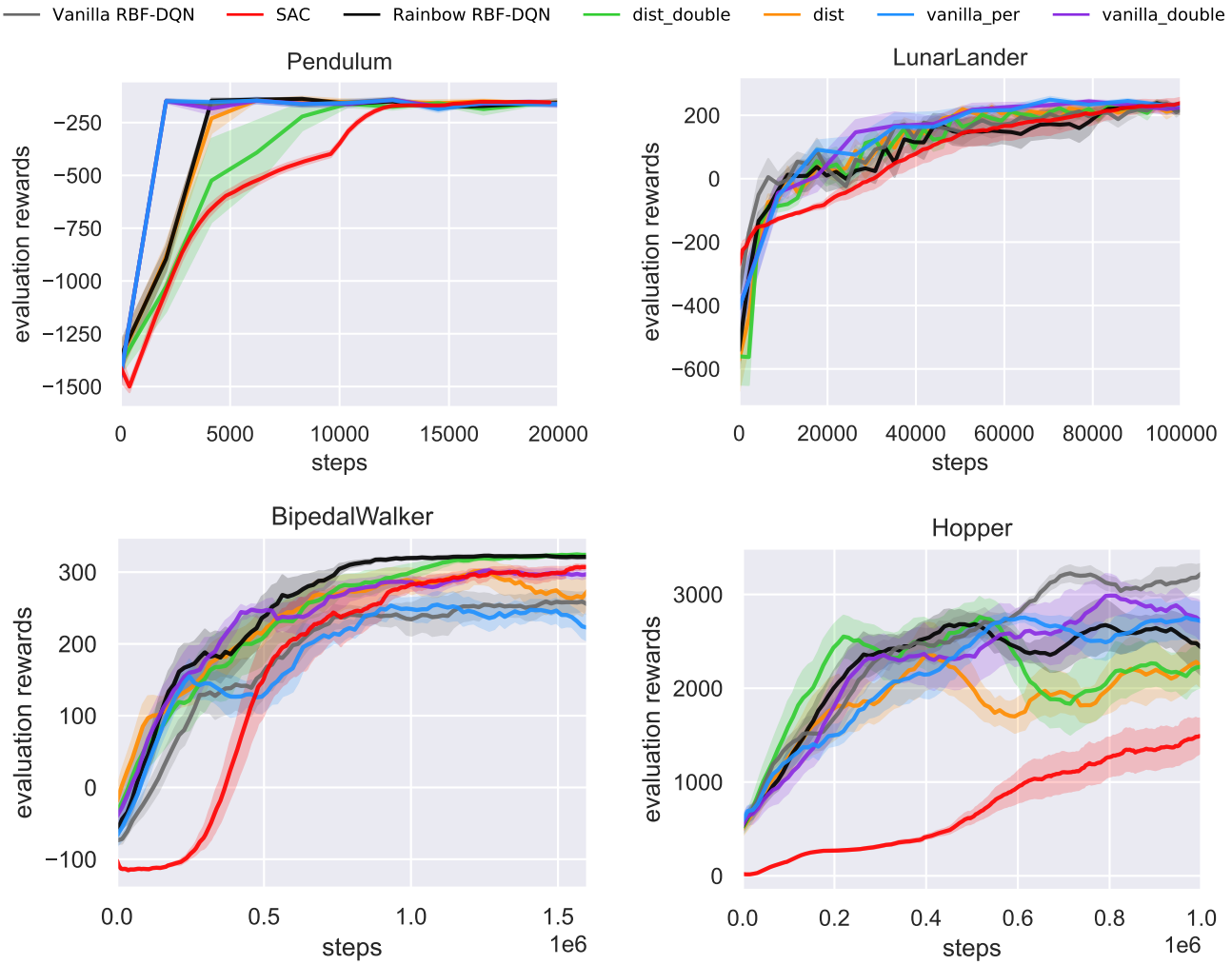

Figure 5: Total evaluation reward per episode for Pendulum, Lunar Lander, Bipedal Walker, and Hopper. All variants ran on 5 seeds and the 95% confidence interval across all seeds is shaded.

SAC. Each task is treated equally, each variation is run for 5 seeds, and evaluation rewards are normalized relative to vanilla RBF-DQN. For full details about how this graph was constructed, see the appendix section A.1.

According to Figure 1, when we average the performance of Rainbow RBF-DQN across all tasks, it is 10% better in final performance than vanilla RBF-DQN and 30% better than SAC (as well as more sample efficient as can be seen by the steeper learning curve). This improvement is in aggregate—on certain tasks like Humanoid, Rainbow RBF-DQN achieves almost 200% the reward of vanilla RBF-DQN, and nearly 25% better than current state of the art methods like SAC. Individual results are given in Figures 5 & 6. On a per-task basis, Rainbow RBF-DQN outperforms SAC on every task except Ant (where it is similar in performance), and only slightly underperforms vanilla RBF-DQN on Hopper. Rainbow RBF-DQN improves performance on Bipedal Walker and improves vanilla RBF-DQN's already strong performance on the HalfCheetah task.

5.2. Ablation Studies

In order to determine the individual contribution of each component of Rainbow RBF-DQN we run ablation experiments, taking one component out of rainbow RBF-DQN to see the impact on the final performance. We also include a few other variations which empirically performed well.

On high dimensional action space tasks, distributional value representations (with quantile regression) for RBF-DQN are crucial to achieving high reward, and are thus the most important ingredient to Rainbow RBF-DQN. In Figure 6, the Humanoid learning curve shows that both vanilla_per (vanilla with PER) and vanilla_double lag behind all the variations with distributional representations included

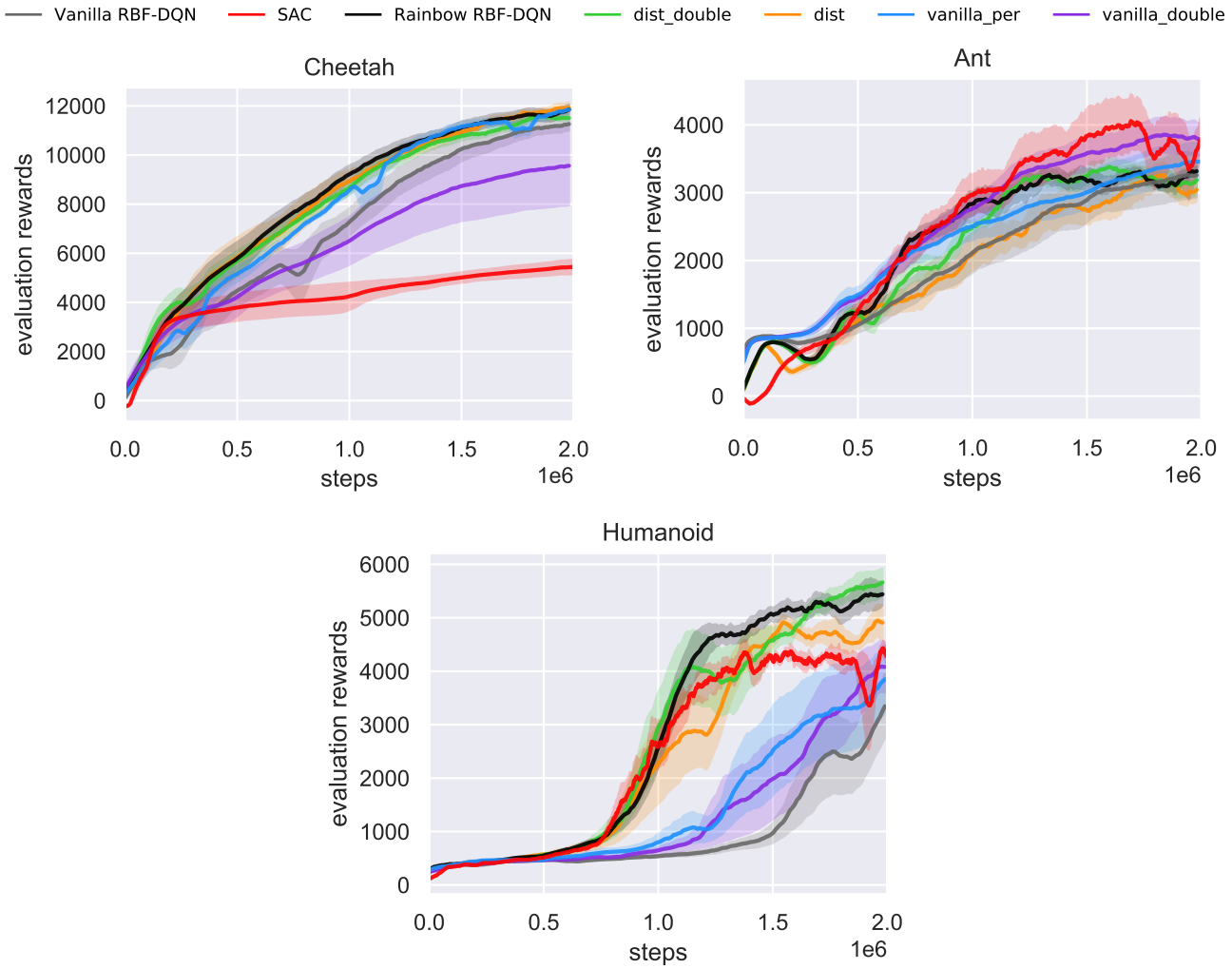

Figure 6: Total evaluation reward per episode for Half Cheetah, Ant, and Humanoid tasks. All variants ran on 5 seeds and the 95% confidence interval across all seeds is shaded.

(dist_double, dist, and Rainbow RBF-DQN). Double networks had only marginal positive impact on Distributional RBF-DQN hinting toward the fact that Distributional alone may have overestimation limiting properties. PER generally speaking led to only marginal improvements when paired with distributional RBF-DQN (we used $\alpha = 0.1$), but we included it because it is a relatively simple addition to the algorithm.

We also implemented and tested ablations of Dueling Networks (Wang et al., 2015), Noisy Networks (Fortunato et al., 2018), and Multi-Step learning (Asis et al., 2018), but did not find that these led to consistent across the board performance increases so we exclude them from the general purpose Rainbow RBF-DQN algorithm. In the interest of space, they are included in the appendix section A.4.

6. Discussion

Results for Dueling Networks, Noisy Networks, and Multi-step returns applied to RBF-DQN can be seen in the appendix section A.4. In this section we review our experience integrating these methods, reasons they may have failed, and potential avenues for future work.

Multi-Step Returns. The poor performance of multi-step returns across these tasks can be explained by the need for an importance sampling correction to account for the difference in likelihood between traversing states under the behavioral and target policies as the replay buffer fills and stale state transitions build up. Due to the deterministic, value based, nature of our policy, it is not entirely obvious how to correct for this bias, which we leave as an area of future exploration. Multi-step returns implemented as is, are incorrect in the off-policy setting because transitions

are highly correlated to the policy they were sampled from. This can make the target Q value incorrect with respect to the current policy and lead to instability in learning, which is something we empirically observed.

Noisy Networks. We found that in order for noisy layers to be effective at exploration in these MuJoCo tasks, they needed to be paired with additional exploration in action space. If noisy layers with normalization were applied *alone* as a replacement for ε-greedy exploration or adding Gaussian noise to the action space, performance drastically decreased, as if no exploration was being performed. We posit this was likely due to exploration in parameter space not translating to significant exploration in action space: performance would flatline. However, the most significant reason why we chose to exclude noisy networks from Rainbow RBF-DQN was due to computational overhead of layer normalization (Ba et al., 2016; Zhang & Sennrich, 2019). Noisy networks show promise, especially on Ant where noisy networks alone with RBF-DQN achieved nearly $4,000$ evaluation reward. Future work will consist of trying to tame the computational overhead while still getting the benefits of increased exploration.

Dueling Networks. Similar to the findings of Hessel et al. (2018), we did not find Dueling Networks to lead to conclusively improved performance across the board. However this could entirely be due to architectural design: unlike the Dueling DQN authors (Wang et al., 2015), we use dueling networks to break up the supports into a base support module (learned by a network) and an offset support module (which produces the offsets required to translate the base support into the supports for each centroid). It could be the case that this architecture is too rigid, overconstraining the agent and hindering the learning process. We leave the exploration of better architectures for dueling networks in Quantile Distributional RBF-DQN as future work.

7. Conclusion

We presented algorithmic extensions for deep value-based learning for the RBF-DQN architecture, and empirically demonstrated improved performance on Humanoid, Bipedal Walker, and Half Cheetah. The most crucial ingredient in Rainbow RBF-DQN is quantile distributional value representations—allowing our agent to excel on high dimensional tasks like Humanoid, outperforming state of the art SAC results. Taken together, our results show that classical DQN enhancements in the discrete action domain can be applied with care to the continuous action domain. We hope Rainbow RBF-DQN becomes one of the go-to value-based methods that RL researchers use for continuous control tasks in high dimensional action spaces.

References

Asadi, K., Parikh, N., Parr, R. E., Konidaris, G. D., and Littman, M. L. Deep Radial-Basis Value Functions for Continuous Control. *arXiv e-prints*, art. arXiv:2002.01883, February 2020.

Asis, K. D., Hernandez-Garcia, J. F., Holland, G. Z., and Sutton, R. S. Multi-step reinforcement learning: A unifying algorithm. In McIlraith, S. A. and Weinberger, K. Q. (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 2902–2909. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16294.

Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 449–458. PMLR, 2017. URL http://proceedings.mlr.press/v70/bellemare17a.html.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv e-prints*, art. arXiv:1606.01540, June 2016.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. Distributional Reinforcement Learning with Quantile Regression. *arXiv e-prints*, art. arXiv:1710.10044, October 2017.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting Fundamentals of Experience Replay. *arXiv e-prints*, art. arXiv:2007.06700, July 2020.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. Noisy

networks for exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rywHCPkAW.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv e-prints*, art. arXiv:1801.01290, January 2018.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In McIlraith, S. A. and Weinberger, K. Q. (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3215–3222. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17204.

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

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

van Hasselt, H. Double q-learning. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pp. 2613–2621. Curran Associates, Inc., 2010.

van Hasselt, H., Guez, A., and Silver, D. Deep Reinforcement Learning with Double Q-learning. *arXiv e-prints*, art. arXiv:1509.06461, September 2015.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1511.06581, November 2015.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8:279–292, 2004.

Zhang, B. and Sennrich, R. Root mean square layer normalization. In *NeurIPS*, 2019.

A. Appendix

A.1. Figure 1 Generating Process

In Figure 1, we compare our integrated agent (rainbow colored) to vanila RBF-DQN (grey) and SAC (red). The data for each task is first normalized using the final performance achieved by vanilla RBF-DQN on that task (each variation on each task is trained for 5 seeds). We then take the mean across tasks to show average performance.

A.2. Related work for Noisy Networks and Multi-Step Returns

Noisy Networks Noisy networks, introduced by Fortunato et al. (2018), provide a method for automatically exploring network parameter space by incorporating learned mean and covariance weights into each layer of the network. To do so, sampled noise is injected at every noisy layer during training. The injected noise is scaled by the current values of the covariance matrix, which results in network parameter exploration. When applied to Q-Learning networks, the effect is that the Q-values will be perturbed, resulting in a stochastic optimal-action selection when greedy policies are applied.

Muti-Step Learning In multi-step learning, instead of training the Q-value on the basis of a single step of temporal difference error, an n-step temporal difference error is used:

$$\text{TD}_{\text{Error}} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_a Q(s_{t+n}, a) - Q(s_t, a_t). \tag{8}$$

When multi-step learning is utilized in conjunction with off-policy learning, the relative probabilities of trajectories produced between current and old policies need to be accounted for by importance sampling. However, based on the work of Fedus et al. (2020), despite the fact that multi-step's reward is based on the agent's current policy during exploration, in practice, multi-step can sometimes give better performance compared to the single-step counterpart.

A.3. Dueling Network Extension for RBF-DQN

In the context of RBF-DQN, we apply dueling networks to the centroid values, with two networks estimating base centroid value and centroid advantages respectively. Because the Q-value estimation for a given state and action is the weighted average of each centroid value (weight determined by distance to a given action based on a kernel), we propose that decoupling the centroid-value into a base value network, V_{base}, and advantage network, A, can improve centroid value estimation. We empirically experimented with different operators in the centroid-value aggregator module below, and found max to be better than other options like mean. Both the centroid value and centroid advantage modules are trained jointly, each with their own optimizer and learning rates.

Each centroid value $v_i(s)$ is then a superposition of the base value with the centroid advantage value, which results in the following interpretation of $Q(s,a) = A(s,a) + V(s) - \max_a(A(s,a))$ values:

$$v_i(s) = V_{\text{base}}(s) + A_i(s) - \max_i(A_i(s)) \tag{9}$$

$$\begin{aligned} Q(s,a) = & \frac{\sum_{i=1}^{N} A_i(s;\phi) e^{-\beta||a-a_i(s;\theta)||}}{\sum_{i=1}^{N} e^{-\beta||a-a_i(s;\theta)||}} \\ & + V_{\text{base}}(s) - \max_i(A_i(s;\phi)) \end{aligned} \tag{10}$$

A.4. Ablation Results for Dueling Networks, Multi-Step Returns, and Noisy Networks with RBF-DQN

Shown in Figures 8 and 9.

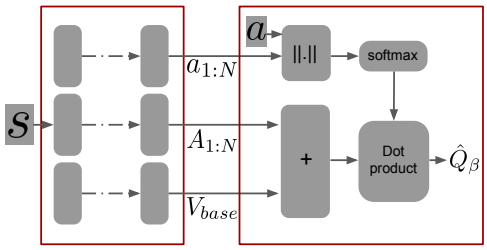

Figure 7: The Dueling RBF-DQN Network Architecture. The plus sign signifies the aggregator module.

A.5. Hyper-parameters for all tasks

The table below lists the hyper-parameters used across all tasks for Rainbow RBF-DQN. We inherit the core hyper-parameters from RBF-DQN (Asadi et al., 2020), including new hyperparameters for the Rainbow components. For the sake of completeness they are listed here and can also be found in the **rbf_hyper_parameters** folder in the provided code.

Table 1: Base Hyper-parameters

Hyper-parameter	value
# hidden layers in centroid base value module	3
# hidden layers in centroid advantage module	3
# hidden layers in centroid location module	1
Replay Buffer Size	$500,000$
gamma	0.99
target network learning rate	0.05
# distributional quantiles	200
optimizer	Adam
# network updates per step	1
batch size	256
PER α	0.1
PER β	scheduled from 0.4 to 1
policy type	ϵ greedy
ϵ greedy exploration reduction exponential	2.75

Each task inherits the hyper parameters above, any changes specific to each task are listed below.

Table 2: Pendulum Hyper-parameters

Hyper-parameter	value
training steps	$40,000$
RBF-DQN # centroids used	100
RBF-DQN temperature	1
learning rate	0.00025
reward clip	20

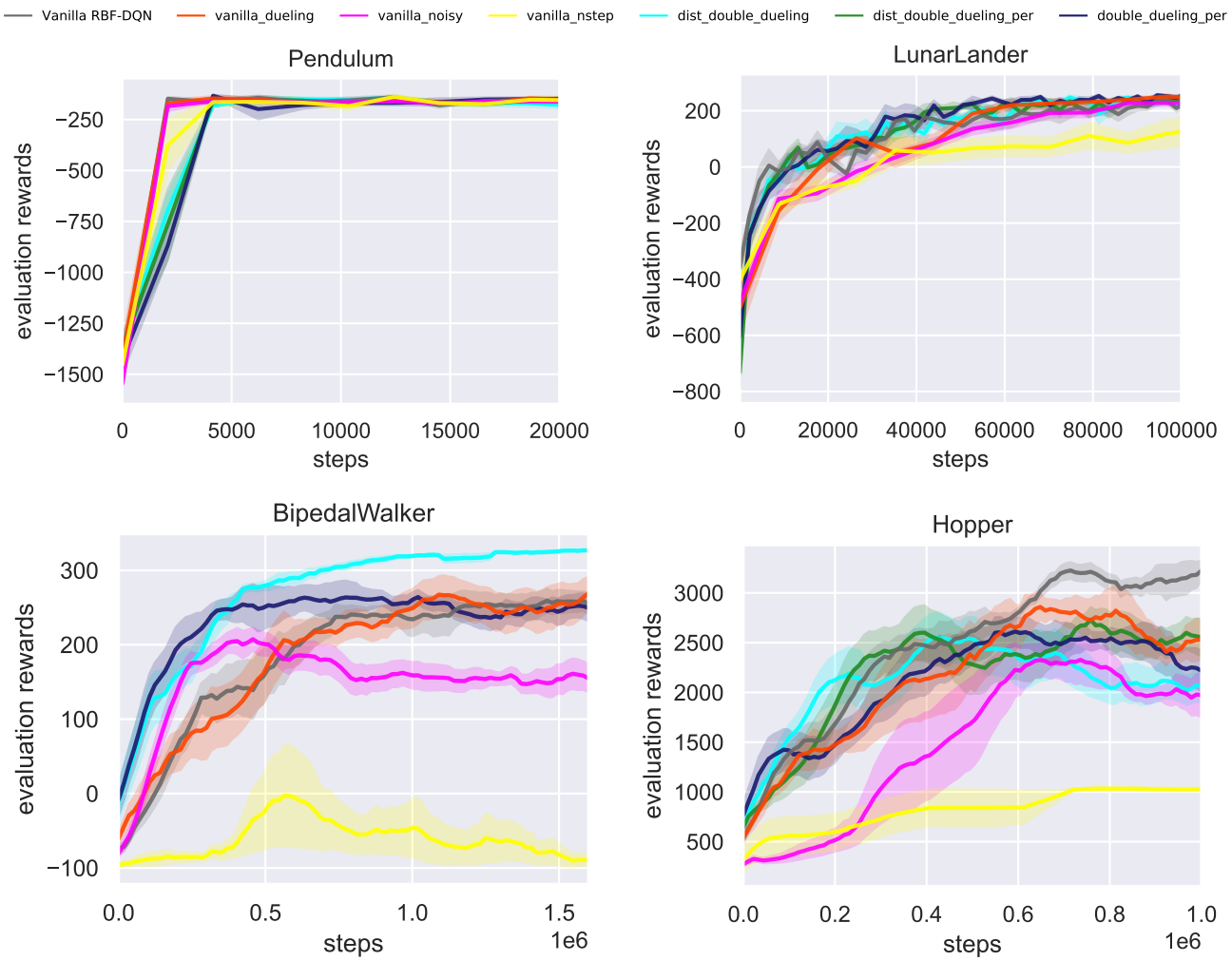

Figure 8: Additional ablation experiments with RBF-DQN, incorporating Dueling Networks, Multi-Step Returns, and Noisy Networks

Table 3: Lunar Lander Hyper-parameters

Hyper-parameter	value
training steps	$200,000$
RBF-DQN # centroids used	100
RBF-DQN temperature	2
learning rate	0.000025
reward clip	20

Table 4: Bipedal Walker Hyper-parameters

Hyper-parameter	value
training steps	$1,600,000$
RBF-DQN # centroids used	100
RBF-DQN temperature	2
learning rate	0.00001
reward clip	20

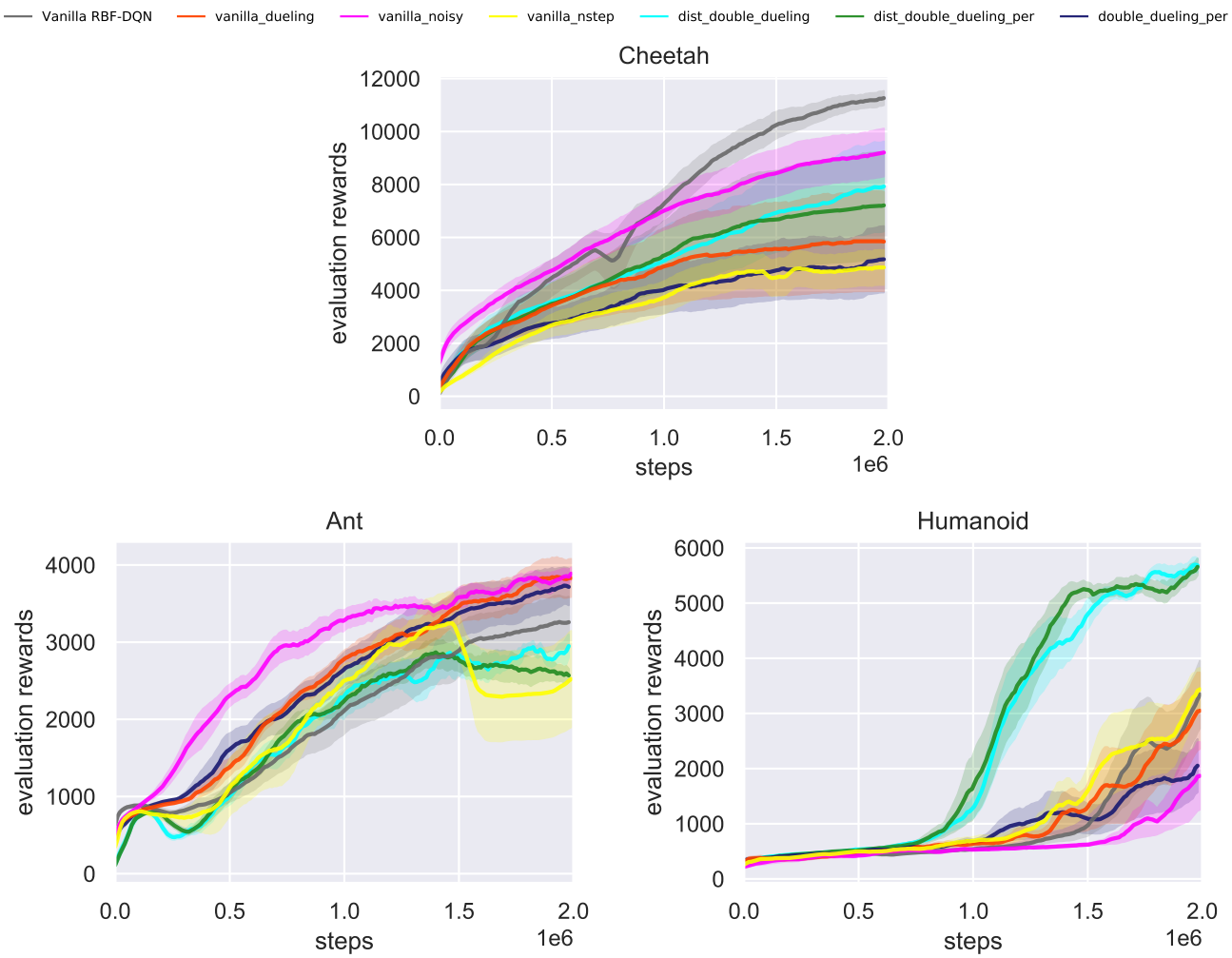

Figure 9: Additional ablation experiments with RBF-DQN, incorporating Dueling Networks, Multi-Step Returns, and Noisy Networks

Table 5: Hopper Hyper-parameters

Hyper-parameter	value
training steps	$1,000,000$
RBF-DQN # centroids used	100
RBF-DQN temperature	0.15
learning rate	0.00005
reward clip	50

Table 6: Half Cheetah Hyper-parameters

Hyper-parameter	value
training steps	$2,000,000$
RBF-DQN # centroids used	500
RBF-DQN temperature	0.25
learning rate	0.0001
reward clip	20

Table 7: Ant Hyper-parameters

Hyper-parameter	value
Replay Buffer Size	$250,000$
training steps	$2,000,000$
RBF-DQN # centroids used	100
RBF-DQN temperature	0.10
learning rate	0.00001
reward clip	20

Table 8: Humanoid Hyper-parameters

Hyper-parameter	value
training steps	$2,000,000$
RBF-DQN # centroids used	500
RBF-DQN temperature	0.10
learning rate	0.00001
reward clip	20

Table 9: Evaluation Frequency for each task

Task Name	Evaluate Agent every # steps
Pendulum-v0	200
LunarLanderContinuous-v2	200
BipedalWalker-v3	1600
Hopper-v3	1000
HalfCheetah-v3	1000
Ant-v3	1000
Humanoid-v2	1000