# Exploring Improvements in Value Based Deep Reinforcement Learning for Continuous Control

**Sreehari Rammohan*, Bowen He*, Shangqun Yu*, Eric Hsiung*, Eric Rosen, George Konidaris**

Intelligent Robot Lab, Brown University

## Abstract

Deep reinforcement learning has been extensively studied, resulting in several extensions that improve its performance, such as replay buffer sampling strategies, distributional value representations, and multi-step bootstrapping targets. 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 applying them to value-based deep reinforcement learning for continuous action spaces. We investigated applying the 6 different DQN extensions in the Rainbow agent to RBF-DQN, a deep valued-based model for continuous control, and experimentally validated their performance on 8 OpenAI Gym and Mujoco domains. We found that naive applications of these extensions to RBF-DQN fail to perform well in some cases, and therefore adapt them to explicitly account for the nature of value-based learning for continuous control. Certain algorithmic improvements to RBF-DQN lead to improved performance on a range of tasks.

## Introduction

Deep Reinforcement Learning (DRL) algorithms have demonstrated success in learning complex policies for sequential 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), bootstrapping with multi-step rewards (Meng, Gorbet, and Kulić 2021), 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, Dueling Networks, Multi-Step Learning, Distributional Reinforcement Learning, and Noisy Nets), and demonstrated that an integrated agent can achieve high performance across 57 Atari games.

*These authors contributed equally.



However, deep value-based approaches have been primarily applied to domains with discrete actions. The continuous control case has largely been dominated by actor-critic methods, until the recent introduction of RBF-DQN (Asadi et al. 2020), a new value-based method applicable to continuous control. While RBF-DQN achieved state-of-the-art results, it does not include any of the extensions included in Rainbow-DQN.

We present an empirical study evaluating the performance of the six algorithmic augmentations included in Rainbow DQN (Hessel et al. 2018) into RBF-DQN (Asadi et al. 2020). We find that applying some of these extensions naively can hurt performance, and we therefore design new versions of them for the continuous control setting. We empirically evaluate each extension in 8 different Mujoco domains, and show improved performance over RBF-DQN in 5 of them (since Lunar Lander and Pendulum are trivial for most agents to solve, they were not counted).

## Background and Related Work

Deep value-based methods use deep learning to approximate a set of parameters $\theta$ to represent 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 (such as the optimal policy) . (Mnih et al. 2013) introduced the first set of improvements, called DQN, that enabled deep neural networks to effectively learn by using the TD-Error objective, which leveraged a replay buffer to store previously experienced transitions for off-policy updates 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 a nearly infinite search space, and employing improvements like discretizing the action space may produce sub-optimal solutions.

Asadi et al. (2020) introduced RBF-DQN, a value-based deep reinforcement learning approach that uses radial-basis functions to approximate the $Q$ function for a continuous action space problem. RBF-DQN predicts both the radial-basis centroids (which represent sampled actions) and their

values, and uses a kernel to efficiently compute the action that can maximize the $Q$-value with bounded error. Specifically, RBF-DQN approximates $Q^*(s,a)$ by optimizing centroid locations $a_i(s;\theta)$ and centroid values $v_i(s;\theta)$ as functions of state $s$ and parameters $\theta$ and $\beta$ with the following equation:

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

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 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 $\beta$ can be tuned to ensure an upper bound on error (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.

## 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 have been termed Rainbow DQN Hessel et al. (2018), which combines double Q-learning (van Hasselt, Guez, and Silver 2015), prioritized experience replay (Schaul et al. 2015), dueling networks (Wang et al. 2015), multi-step learning (Asis et al. 2018), distributional RL (Bellemare, Dabney, and Munos 2017), and noisy networks (Fortunato et al. 2018). In this section, we 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. The Double Q-learning networks use two estimators to address the overestimation bias. Double-DQN (van Hasselt 2010) uses an online network, parameterized by $\theta$, to pick the action that is evaluated by the target network, parameterized by $\theta^-$. 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 functional approximators (van Hasselt, Guez, and Silver 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 $A(s,a)$ (because $Q(s,a) = V(s) + A(s,a)$). Each stream of computation shares the same initial layers for processing the input (parameterized by $\theta$), but they then split into two separate sequences of neural network layers that compute estimates of the state value (parameterized by $\beta$) and advantage values (parameterized by $\alpha$) separately. The two streams are then combined to produce a single Q function estimate using the following equation:

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

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, get replayed more frequently. There are two important hyperparameters: $\alpha$, to what extent priority is incorporated into the vanilla uniform sampling process, and $\beta$, the compensation factor for performing importance sampling on the TD-error for the transitions in the buffer.

**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 one single step of temporal difference error, a $n$-step temporal difference error is used:

$$\text{TD}_{\text{Error}} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_a Q(s_{t+n},a) - Q(s_t,a_t). \quad (5)$$

When multi-step learning is utilized in conjunction with off-policy learning, the relative probabilities of trajectories produced between current and old policies needs 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.

**Distributional Q-Learning** Bellemare, Dabney, and Munos (2017) proposed learning a distribution over $Q$ values instead of learning a single expected value. They discretized the output value range from $v_{min}$ to $v_{max}$ using 51 supports for the distribution on Atari game environments. Cross Entropy Loss is then used to compute the distance between the predicted and target distributions to update the network:

$$-\sum_{i=1}^{N} p_i \log q_i, \tag{6}$$

where $p_i$ is the probability at the support $i$ in the predicted distribution, and $q_i$ is the probability for the target distribution. The distributional agent, called C51, outperformed vanilla DQN on certain Atari tasks.

## Extensions to RBF-DQN

In this work, 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 to RBF-DQN without modification.

### Double Q-Learning

We hypothesize using two estimators can also prevent RBF-DQN from over estimating action values.

To adapt double Q-learning networks for RBF-DQN, the optimal action is selected using the online centroid and centroid value modules, $Q(s', a; \theta)$, $a_i(s'; \theta)$, $v_i(s'; \theta)$, and then producing a Q-value using the centroid and centroid values from the target network $Q(s', a; \theta^-)$, $a_i(s', \theta^-)$, $v_i(s', \theta^-)$ respectively.

$$Q(s', a^*; \theta^-) = \frac{\sum_{i=1}^{N} e^{-\beta||a^* - a_i(s';\theta^-)||} v_i(s'; \theta^-)}{\sum_{i=1}^{N} e^{-\beta||a^* - a_i(s';\theta^-)||}}, \tag{7}$$

where

$$a^* = \underset{a_i}{\operatorname{argmax}}\, Q(s', a_i; \theta). \tag{8}$$

The action $a^*$ is the centroid location associated with the highest $Q$ value (computed with equation 1).

### Dueling Networks

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, $V$, 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. Because our state is a low-dimensional vector representation of the environment, no shared feature extraction module is needed—both modules take in the same state.

$$v_i = V(s) + A(s) - \max(A(s)). \tag{9}$$

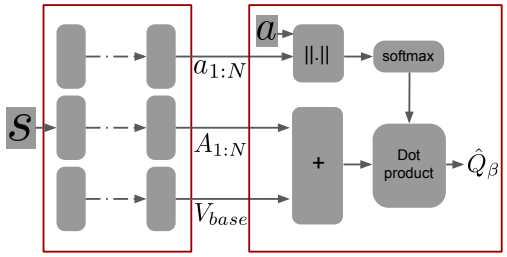


Figure 1: Dueling RBF-DQN Network Architecture. The plus sign signifies the aggregator module

### Distributional Q-Learning

In the RBF-DQN setting, the network's value module is modified by adopting the distribution representation. During the forward pass, the centroid value is given by the expectation of the distribution. The radial basis weights are kept to merge the distributions through support-wise multiplication:

$$[w_1 \quad w_2 \quad ... \quad w_m] \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn}, \end{bmatrix} \tag{10}$$

where $m$ represents the number of centroids, and $n$ represents the number of supports to the discrete distribution.

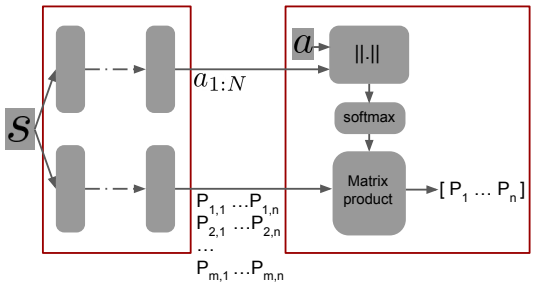


Figure 2: Distributional RBF-DQN Network Architecture

The predicted distribution is updated toward the target one, which is related to the distribution of the best centroid at $s'$ in terms of the expectation, discounted by $\gamma$ and added by the immediate reward at state $s$.

## Experiments

We performed ablation experiments using the methods described in section 4 on 8 different continuous control domains from the OpenAI Gym suite (Brockman et al. 2016):

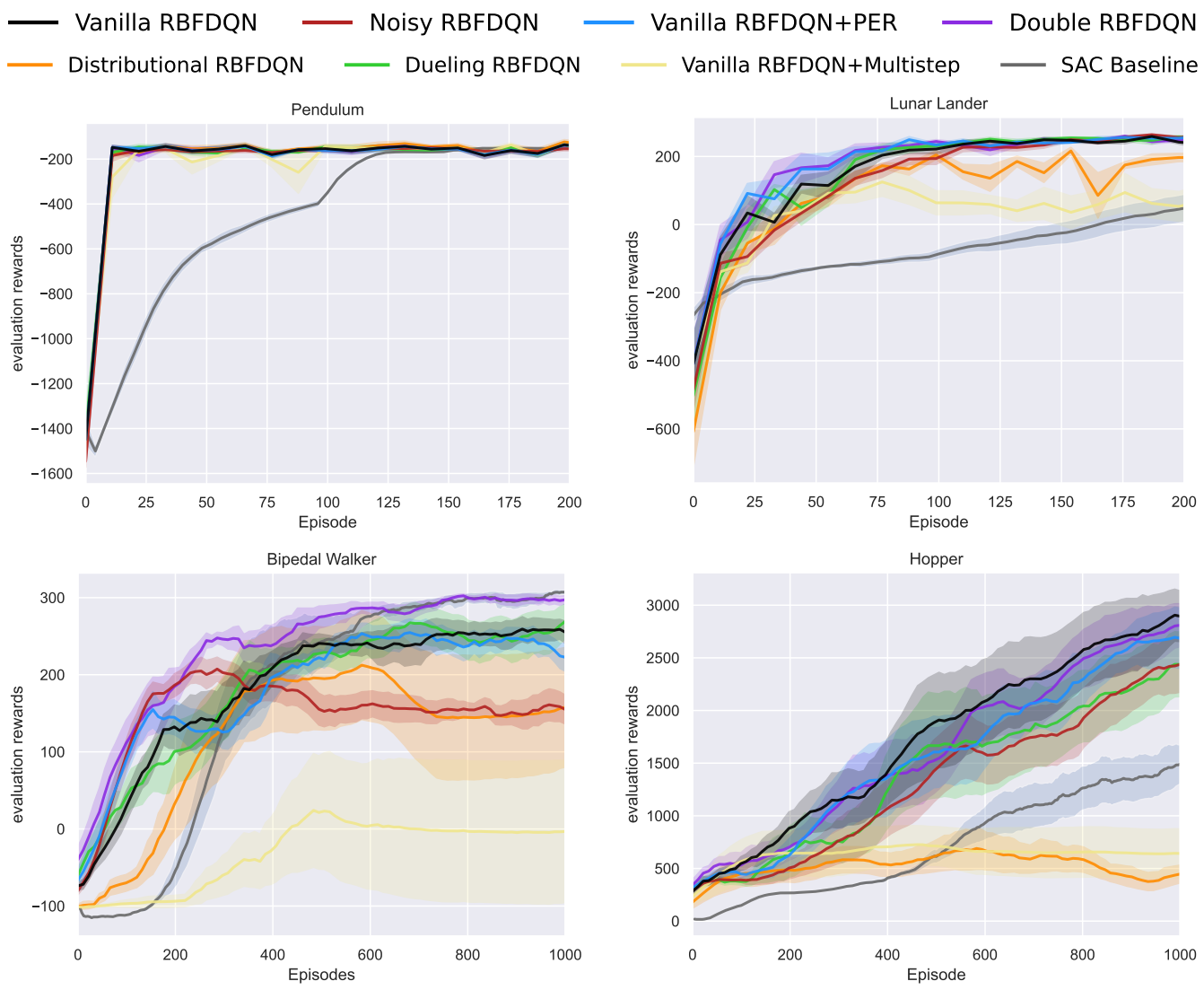


Figure 3: Total evaluation reward per episode for Pendulum, Lunar Lander, Bipedal Walker, and Hopper. All variants ran on 5 seeds and the 95% confidence interval across all seeds is shaded.

Pendulum, Lunar Lander, Bipedal Walker, Hopper, Half Cheetah, Ant, Humanoid, and Walker2D. For all these domains, we used the low dimensional state space and the default dense reward specifications. Because RBF-DQN, and deep RL in general, can show high sensitivity to hyperparameter configurations, we ran extensive parameter sweeps to ensure our benchmarks reflected the highest possible performance for each variation.

We empirically found that PER is extremely sensitive to the hyperparameters $\alpha$ (the degree to which prioritization is used) and $\beta$ ( sampling correction coefficient). For RBF-DQN with PER, performance was generally best with $\beta$ annealed from $0.4$ at the start of training to $1.0$ by the end of training and $\alpha$ set to $0.1$. In the case of applying Noisy Networks to RBF-DQN, all linear layers were replaced with noisy layers with layer normalization (Ba, Kiros, and Hinton 2016) applied, and initial covariance settings set at $\sigma \in \{0.25, 0.5, 1.0, 2.0, 10.0\}$. We found that setting $\sigma = 0.25$ worked well for all tasks except for Hopper and Half-Cheetah, in which $\sigma = 1.0$ or $\sigma = 2.0$ yielded better performance. We implemented multi-step reward with a short horizon of 2 to 4 steps due to the continuous nature of the tasks evaluated.

For exploration, we applied $\varepsilon$-greedy exploration to all tasks except for Ant, Humanoid, and Walker2D, where Gaussian noise was added with noise between $0.10$ and $0.25$. Pendulum, and Lunar Lander were trained for $200$ episodes, Bipedal Walker, Hopper, and Walker2D were trained for $1000$ episodes and Humanoid, Half Cheetah and Ant were trained for $2000$ episodes.

Our results are given in in Figures 3 and 4. For each domain, we report the cumulative evaluation returns for our

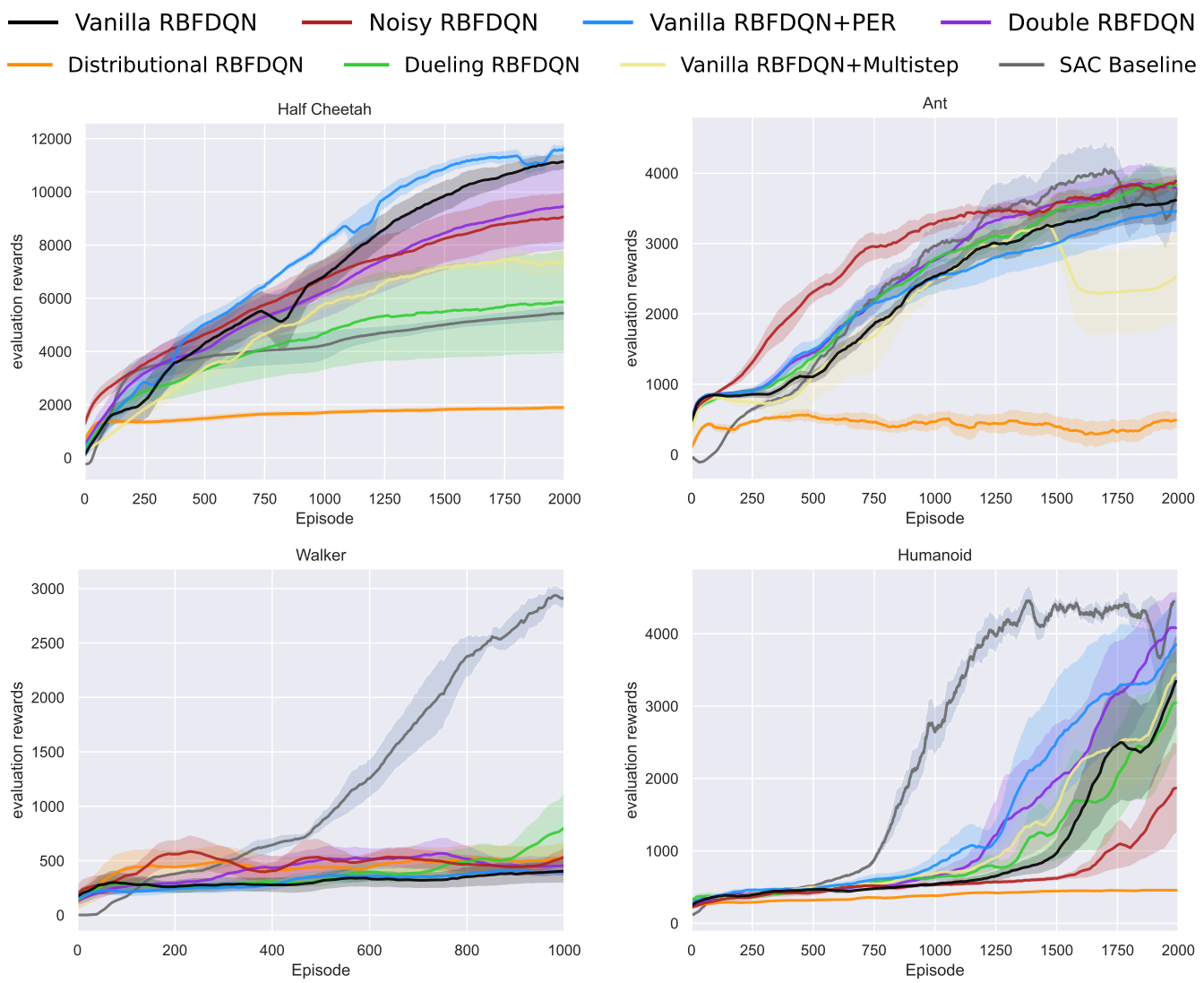


Figure 4: Total evaluation reward per episode for Half Cheetah, Ant, Walker, and Humanoid tasks. All variants ran on 5 seeds and the 95% confidence interval across all seeds is shaded.

test agent vs. the number of training episodes. Except in the case of Hopper, where we match baseline vanilla RBF-DQN performance, at least one of the variations presented in this work is able to outperform vanilla RBF-DQN performance in each task. To provide context for these methods in comparison to other state of the art agents commonly employed, we ran SAC (Haarnoja et al. 2018) on all tasks (5 seeds).

In order for noisy layers to be effective at exploration in these Mujoco tasks, we found they needed to be paired with additional exploration in action space: if noisy layers with normalization were applied *alone* as a replacement for $\varepsilon$-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. This was verified by adding $\varepsilon$-greedy or Gaussian exploration back in and seeing learning progress as one would expect.

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.

Looking at Humanoid, the Double and PER variations are able to outperform vanilla RBF-DQN by a significant amount. Furthermore, Double and Noisy Network variations can help RBF-DQN converge to better policies faster, as seen by the steeper learning curves for Bipedal Walker and

Ant respectively.

## Conclusion

We extended multiple discrete action space methods to work in the continuous action space, improving the performance of RBF-DQN on 5 out of 8 tasks. The enhancement methods designed for classical DQN can be augmented to fit this model, with varying degrees of success. Double Q-Learning has been shown empirically to almost always lead to improvements in task performance due to the fact that it solves for over estimation bias. Multi-step bootstrapping surprisingly hurts performance, most likely due to unaccounted for importance sampling bias. Lastly, the current design of distributional RBF-DQN may trap the agent to sub optimal motor control policies. We propose that the interaction between the value module and the centroid module may play a key role to understanding the underlying behavior of different components - providing a path forward to further improve RBF-DQN.

Taken together, our results suggest that classical DQN enhancements in the discrete domain can be cross-applied to the continuous control case.

Future work will investigate the interplay between these 6 methods and how they can be effectively brought together to both speed up learning as well as increase overall agent performance.

## References


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

Asis, K. D.; Hernandez-Garcia, J. F.; Holland, G. Z.; and Sutton, R. S. 2018. 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*, 2902–2909. AAAI Press.

Ba, L. J.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *CoRR*, abs/1607.06450.

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

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. 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*, 449–458. PMLR.

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

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

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. 2018. 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.

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

Hessel, M.; Modayil, J.; van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M. G.; and Silver, D. 2018. 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*, 3215–3222. AAAI Press.

Meng, L.; Gorbet, R.; and Kulić, D. 2021. The effect of multi-step methods on overestimation in deep reinforcement learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 347–353. IEEE.

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

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

van Hasselt, H. 2010. 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*, 2613–2621. Curran Associates, Inc.

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

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