Deep Reinforcement Learning with Value Function Approximation of the Q-Action

José Ridruejo Tuñón, Joaquín Mir Macías & Francisco Javier Ríos
Deep Reinforcement Learning

ABSTRACT
This work investigates Deep Reinforcement Learning (DRL) for visually rich continuous control using MuJoCo tasks from the DeepMind Control Suite accessed via Gymnasium. We first implement a Deep Q-Network (DQN) baseline that learns a visual state representation and an approximate Q-function from RGB observations over a discretized action space. We then extend this baseline to Rainbow DQN, incorporating different improvements. Through experiments on locomotion and obstacle-avoidance tasks, we compare DQN and Rainbow in terms of sample efficiency, learning stability, and exploration, and conduct an ablation study on selected Rainbow components. Our results highlight when advanced DQN variants offer clear benefits over standard DQN in complex visual control environments.

Key words: Deep Reinforcement Learning, Deep Q-Network, Rainbow DQN, MuJoCo, DeepMind Control Suite, Gymnasium, Visual Control, Continuous Control, Locomotion, Obstacle Avoidance

Repository: `https://github.com/pepert03/DQN-Rainbow-Pixel-Control`

1 INTRODUCTION

Deep Reinforcement Learning (DRL) deals with high-dimensional state spaces, such as images, which are not feasibly handled with traditional RL methods. This project will revolve around the implementation and analysis of Deep Q-Networks (DQN) and Rainbow DQN, two prominent DRL algorithms that leverage deep neural networks to approximate the action-value function. These will be applied to continuous control tasks in visually rich environments, which deal with the challenge of learning effective policies from raw pixel observations. Specifically, the tasks will be related to locomotion and obstacle avoidance, where agents must learn to move forward while avoiding collisions.

1.1 Deep Q-Networks Fundamentals

In order to understand the implementation of DQN and Rainbow DQN, we must first go through a slim review on the fundamentals of Q-learning. Traditional tabular Q-learning introduces an MDP (Markov Decision Process) where an agent learns to take actions in an environment to maximize cumulative rewards. The Q-function, denoted as $Q(s, a)$, represents the expected return of taking action a in state s and following the optimal policy thereafter. The update rule for Q-learning is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) ^{\star}$$

where α is the learning rate, γ is the discount factor, r is the reward received after taking action a in state s, and s' is the next state.

However, in high-dimensional state spaces, such as those involving images, it is infeasible to maintain a Q-table (due to the exponential growth of state-action pairs). This is where Deep Q-Networks come into play, using a neural network to approximate the Q-function. The DQN algorithm (Mnih et al., 2015) thus replaces the Q-table with a function approximator, typically a convolutional neural network (CNN) for processing visual inputs. This approximator, denoted as $Q_\theta(s, a)$, is trained (tuning its parameters θ) to minimize the temporal difference error:

$$L(\theta) = \mathbb{E}_{s,a,r,s'} \left[\left(r + \gamma \max_{a'} Q_{\theta^-}(s',a') - Q_\theta(s,a) \right)^2 \right]$$

*Typically, the learning rate α is often decayed over time to ensure convergence, and the discount factor γ is chosen based on the desired balance between immediate and future rewards.

where θ^- are the parameters of a target network that is periodically updated to stabilize training.

The DQN algorithm incorporates two stabilizing techniques:

- **Experience Replay:** Instead of learning from consecutive samples, DQN stores past experiences in a replay buffer and samples mini-batches randomly for training:

$$\mathcal{D} = \{(s, a, r, s')\}$$

This breaks the correlation between consecutive samples and improves data efficiency.

- **Target Network:** A separate target network with parameters θ^- is used to compute the target Q-values, which is updated less frequently than the main network. This helps to stabilize learning by providing a more consistent target for the Q-value updates. In order to do so, the target is defined as:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

and the network parameters θ are updated to minimize the loss $L(\theta) = (y - Q_\theta(s, a))^2$.

These improvements allowed DQN to successfully learn policies from high-dimensional inputs, such as raw pixel observations in Atari games. However, DQN still suffers from issues like overestimation bias and sample inefficiency, which led to the development of Rainbow DQN.

1.2 Rainbow DQN

As we have briefly mentioned in the previous section DQN, while being a significant advancement in DRL, suffered from several limitations:

- **Overestimation Bias:** The max operator in the target calculation can lead to overestimation of action values, which can destabilize learning.
- **Sample Inefficiency:** Since all replayed transitions are treated equally, DQN may waste time learning from unimportant experiences.
- **Lack of Exploration:** DQN typically uses ϵ-greedy exploration, which is often suboptimal and unstable (Mnih et al., 2015).

To address these issues, Rainbow DQN (Hessel et al., 2018) was proposed as a way to combine several previously independent improvements into a single, stronger DQN variant. The key components of Rainbow DQN include:

- **Double Q-learning:** (van Hasselt et al., 2016) This technique decouples action selection from action evaluation to reduce overestimation bias. The online network is used to select the action via $a' = \arg\max_{a'} Q_\theta(s', a')$, while the target network is used to evaluate the action's value:

$$y = r + \gamma\, Q_{\theta^-}(s', a')$$

- **Prioritized Experience Replay:** (Schaul et al., 2016) Instead of sampling transitions uniformly from the replay buffer, this method prioritizes transitions that have a higher temporal difference (TD) error, which are more informative for learning the value function (since they provide more of a surprise to the agent)
- **Dueling Architecture:** (Wang et al., 2016) This architecture decomposes the Q-function into two separate estimators: one for the state value function $V(s)$ and another for the advantage function $A(s, a)$. The Q-value is then computed as:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$$

This allows the network to discern between different actions which may have similar values in certain states, improving learning efficiency.

- **n-step Returns:** This technique uses multi-step instead of simple one-step returns to provide a richer learning signal, which can speed up learning by propagating rewards more quickly through the state space.
- **Distributional RL:** Instead of estimating the expected return, this method models the entire distribution of returns, which can capture more information about the uncertainty and variability in the environment.
- **Noisy Networks:** This approach replaces the ϵ-greedy exploration strategy with a parameterized noise added to the network weights, which enables an adaptive exploration strategy that typically leads to better performance and stability.

As we will replicate in further sections, the authors also performed an ablation study to evaluate the contribution of each component, demonstrating that the combination of these techniques leads to significant performance improvements over the original

DQN across a variety of Atari games. Thus, one of the main goals of this project will be to implement Rainbow DQN and perform a similar ablation study to understand the impact of each component in the context of continuous control tasks with visual inputs (studying specially the possible differences with respect to the authors' results).

1.3 Environments: MuJoCo and Deepmind Control Suite

It would be possible to implement DQN and Rainbow DQN in a variety of environments, as well as (given a proper budget) to create our own custom environments. However, for the sake of simplicity and reproducibility, we will focus on some simulation engines that are widely used in the DRL community. Specifically, we will use MuJoCo (Todorov et al., 2012), which stands for *Multi-Joint dynamics with Contact*, a physics engine designed for research in robotics and control. It provides accurate and efficient simulation of complex articulated bodies with contact, collision and friction dynamics. What's more, it supports a myriad of systems, such as walkers, humanoids and quadrupeds, which makes it ideal for testing DRL algorithms in continuous control tasks.

To access MuJoCo environments, we will use the DeepMind Control Suite (Tassa et al., 2018), which is a collection of standardized benchmarks for DRL algorithms. It provides a set of continuous control tasks with varying levels of difficulty, which are designed to set a standard for evaluating the performance of DRL algorithms.
In this project, we will focus on the **HalfCheetah** task [†], which involves controlling a two-legged robot to move forward as fast as possible (more on the specific task in the next sections). The reason why we chose this specific task is that it is simple enough for amateurs in MuJoCo and DPM to implement and test their algorithms, while still being complex enough to provide a meaningful challenge for DRL algorithms.

1.4 Environments Framework: Gymnasium

Last, we must introduce the framework we will use to interact with the environments. Gymnasium (Farama Foundation, 2022) is a toolkit for developing and comparing reinforcement learning algorithms (successor of OpenAI Gym). It standardizes the interface for RL environments, providing an API that allows agents to interact with a wide variety of environments in a consistent way. This allows agents to be plugged in and tested across different environments without needing to modify the underlying code.

Therefore, our stack will consist of:

- **Physics Engine:** MuJoCo, which will handle the physics simulation of our environments.
- **Task Layer (Environments):** DeepMind Control Suite, which provides a set of standardized tasks for testing our algorithms.
- **Interface Layer:** Gymnasium, by means of its abstractions, will allow us to interact with the environments in a consistent way, providing methods for resetting the environment, taking actions, and receiving observations and rewards.

[†] It must be noted that DPM exposes either low-dimensional state vectors or visual observations. In order to make the project as close to reality, we will work exclusively with RGB observations.

REFERENCES

Farama Foundation, 2022, Announcing the farama foundation: The future of open source reinforcement learning libraries: `https://farama.org/Announcing-The-Farama-Foundation`. (Accessed: 2026-02-14).

Hessel, M., J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, 2018, Rainbow: Combining improvements in deep reinforcement learning: Proceedings of the AAAI Conference on Artificial Intelligence, **32**.

Mnih, V., K. Kavukcuoglu, D. Silver, et al., 2015, Human-level control through deep reinforcement learning: Nature, **518**, 529–533.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver, 2016, Prioritized experience replay: Presented at the 4th International Conference on Learning Representations.

Tassa, Y., T. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, et al., 2018, Deepmind control suite: arXiv preprint arXiv:1801.00690.

Todorov, E., T. Erez, and Y. Tassa, 2012, MuJoCo: A physics engine for model-based control: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 5026–5033.

van Hasselt, H., A. Guez, and D. Silver, 2016, Deep reinforcement learning with double q-learning: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2094–2100.

Wang, Z., T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, 2016, Dueling network architectures for deep reinforcement learning: Proceedings of the 33rd International Conference on Machine Learning, 1995–2003.