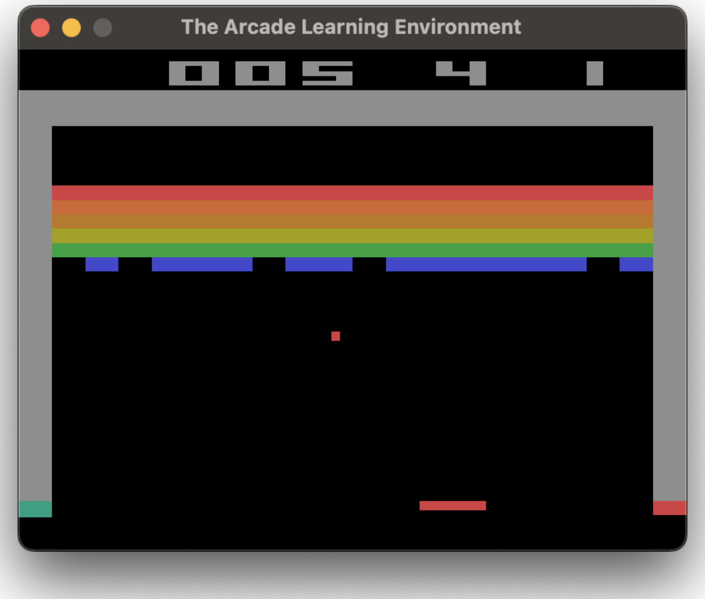
Original DQN Paper Implementation and Upgrades

Tom Genlis - Summer 2024



Abstract

This report presents the re-implementation and enhancement of the Deep Q-Network (DQN) originally introduced by Mnih et al. in 2015. Motivated by a desire to deepen our understanding of reinforcement learning (RL) and deep learning, and driven by curiosity about reimplementing influential papers, we replicated the DQN with modifications to accommodate computational constraints. We then upgraded the model by integrating components from the Rainbow algorithm, including dueling networks, prioritized experience replay, n-step returns, and noisy networks. Comprehensive evaluations were conducted by comparing each version against a random action baseline and analyzing the impact of different parameters and network configurations. Although we could not train the models long enough to achieve the highest scores reported in the original paper due to computational limitations, the learning curves indicate that with extended training, especially for the Rainbow-enhanced DQN, the models would continue to improve. This work provides insights into the practical implementation of advanced RL techniques and underscores the educational value of reimplementing foundational research.

1 Introduction

1.1 Background

Deep Reinforcement Learning (DRL) combines Reinforcement Learning (RL) with deep neural networks to enable agents to learn optimal behaviors in complex environments. This synergy has led to breakthroughs in areas such as game playing, robotics, and autonomous systems.

1.2 Overview of DQN

The Deep Q-Network (DQN) introduced by Mnih et al. in 2015 achieved human-level performance on Atari 2600 games using raw pixel inputs. Key innovations included experience replay and target networks, which stabilized training.

1.3 Motivation

Our primary motivation for this project was to deepen our understanding of reinforcement learning and deep learning by engaging directly with seminal research in the field. Driven by curiosity about the practical aspects of reimplementing influential papers, we chose to replicate the original DQN to explore its foundational mechanics firsthand. This hands-on approach allowed us to gain valuable insights into the intricacies of RL algorithms and the challenges of implementing them. Furthermore, by incrementally enhancing the DQN with components from the Rainbow algorithm, we aimed to evaluate the individual and combined effects of these upgrades on performance. This process not only satisfied our intellectual curiosity but also contributed to our educational growth in advanced RL techniques.

1.4 Baseline Comparison

Comparing the agent's performance to a random action baseline provides a fundamental benchmark. It ensures that improvements are meaningful and not due to environmental dynamics.

1.5 Contributions

Our efforts included re-implementing the DQN with modifications to suit computational resources and integrating Rainbow components such as dueling networks, prioritized experience replay, n-step returns, and noisy networks. We conducted a comprehensive evaluation by comparing each version against a random action baseline and analyzing parameter impacts. Finally, we presented graphs illustrating training performance enhancements to benchmark the improvements.

2 DQN Implementation with Modifications

2.1 Implementation Details

Network Architecture: The network has **1,686,180** trainable parameters, close the paper implementation.

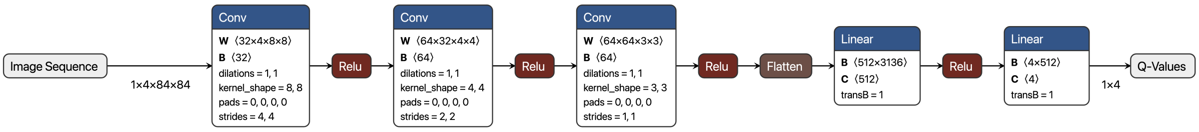


Figure 1: DQN Network Architecture

Hyperparameters:

Parameter	Value
Loss Function	Huber loss
Learning Rate	0.0002
Optimizer	Adam optimizer
Discount Factor (γ)	0.99
Replay Memory Size	150,000
Batch Size	32
Target Network Update Frequency	Every 1,200 steps
Frame Skip	4
Epsilon (ϵ) Decay	From 1.0 to 0.1 over first 10% of steps
Total Training Steps	1,500,000
Replay Start Size	50,000

Table 1: Hyperparameters for DQN Implementation

2.2 Experimental Setup

2.2.1 Environment:

We used the Atari 2600 game **Breakout** as our testing environment due to its manageable complexity and the availability of comparative metrics.

2.2.2 Training Protocols and Evaluation Metrics:

The agent was trained for 1,500,000 steps, with the target network updated every 1,200 steps. Frame preprocessing involved grayscale conversion, resizing to 84×84 pixels, normalization, and stacking of frames. We evaluated the agent using metrics such as average reward per episode, loss values, and Q-value estimates.

2.2.3 Testing Protocols:

We conducted 50 episodes to evaluate the agent's performance after training, calculating the average reward per episode to assess effectiveness. During testing, we introduced a 1% chance of random actions, as advised by the original paper, to prevent the agent from getting stuck in repetitive action loops and prevent overfitting to the training environment.

2.2.4 Baseline: Random Actions

We implemented a baseline where actions were selected uniformly at random, providing a performance floor for comparison.

2.3 Results and Analysis

2.3.1 Performance Metrics

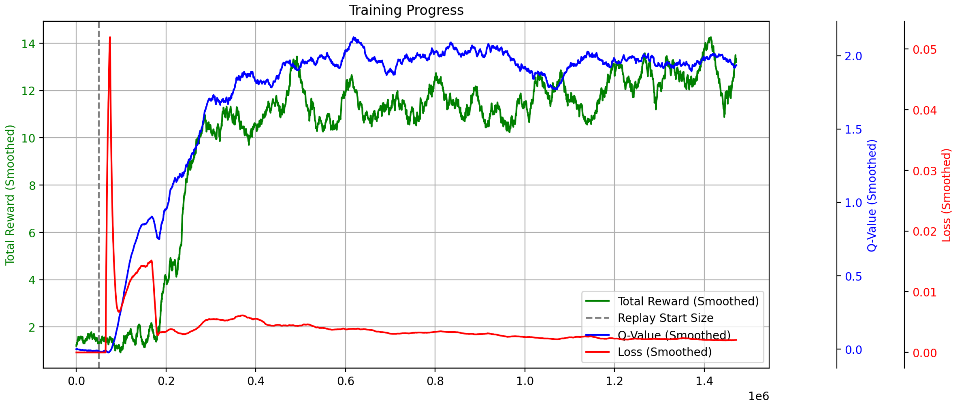


Figure 2: Training Performance of DQN

The agent's average reward increased over time, indicating learning progress. Q-value estimates converged during training, and loss values decreased, suggesting convergence. Although we could not train for as long as in the original paper, the upward trend in the learning curve suggests that with extended training, the agent would continue to improve.

2.3.2 Comparison with Original Paper

Our implementation reached lower performance levels and slower convergence than reported by Mnih et al. Possible reasons include the reduced replay memory size, the use of the Adam optimizer instead of RMSProp, and limited training time due to computational constraints. Despite these differences, the learning curve suggests that with longer training, the agent's performance would continue to improve, potentially matching or approaching the results of the original paper. In Breakout, extended training would likely enable the agent to discover advanced strategies, such as focusing on breaking through the sides to reach the top of the brick wall, leading to higher scores.

3 Upgrades Using Rainbow Components

3.1 Overview

The Rainbow algorithm combines several enhancements to improve DQN performance, including Double DQN, Prioritized Experience Replay, Dueling Network Architecture, Multi-Step Learning, and Noisy Networks.

3.2 Incorporation of Components

We incorporated the **Dueling Network Architecture** by modifying the network to include separate streams for state value and advantage, which are then combined to produce Q-values. The network now has **6,507,690** trainable parameters, which 3.85 times more than the previous model, this can be explained by having two independent pipelines inside the model.

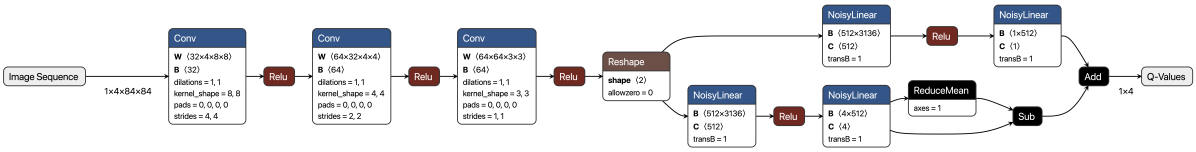


Figure 3: Rainbow DQN Network Architecture

We implemented **Prioritized Experience Replay** by modifying the replay buffer to prioritize experiences with higher TD errors. **N-Step Returns** were incorporated by adjusting target calculations to include rewards over multiple steps. We added **Noisy Networks** by replacing certain layers with noisy layers to facilitate exploration.

3.3 Results and Analysis

3.3.1 Performance Metrics

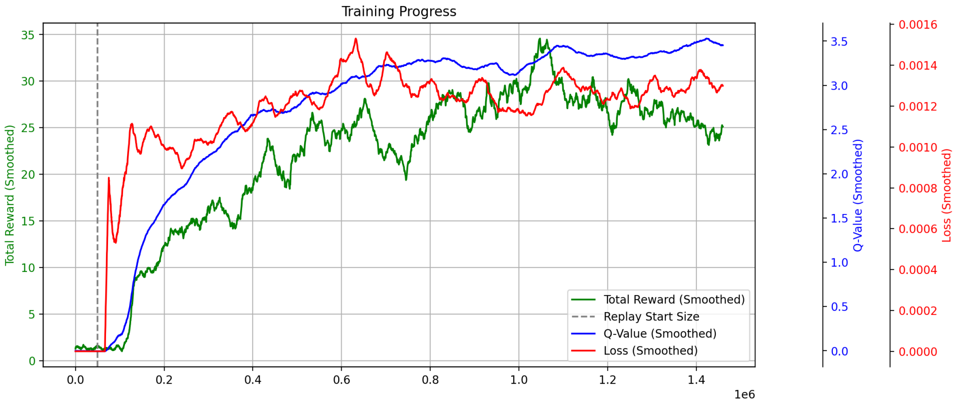


Figure 4: Training Performance of Rainbow DQN

The Rainbow DQN showed improved rewards over the classic DQN and enhanced learning stability due to the integrated components. The upward trajectory of the learning curve indicates that the Rainbow-enhanced DQN is on a promising path toward achieving higher performance levels with extended training.

3.3.2 Comparison with Original Paper

While the Rainbow-enhanced DQN showed performance closer to the results reported by Mnih et al., it was still lower due to the limited training duration. Given the positive trend in the learning curve, we anticipate that with longer training times, the Rainbow DQN would continue to improve and potentially reach higher scores. Extended training would allow the agent to refine its strategies, such as breaking through the sides of the brick wall in Breakout to maximize points.

4 Benchmarking and Evaluation

We aggregated results from all experiments, as shown in the final performance graph.

4.1 Training Results

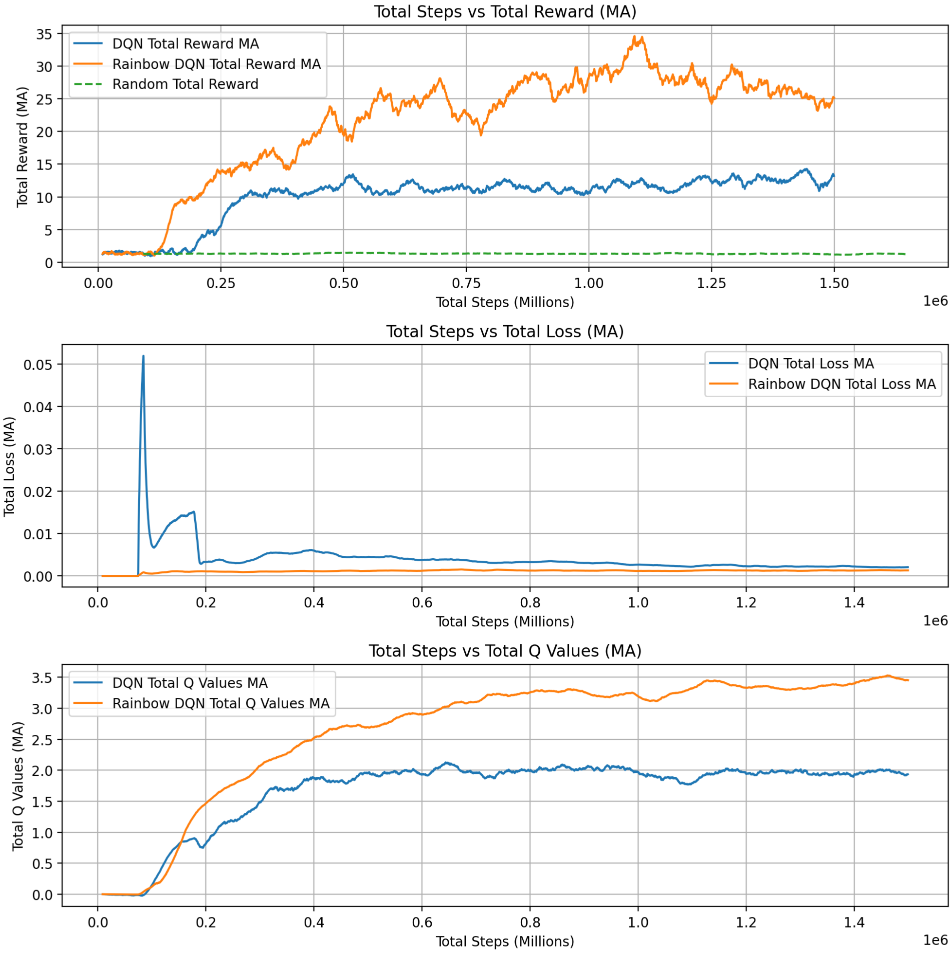


Figure 5: Performance Comparison of Agents

The Rainbow DQN outperformed both the classic DQN and the random baseline.

4.2 Testing Results

Experiment	r_{avg}	r_{σ}	r_{max}	r_{min}
Random Baseline	1.40	1.15	5.00	0.00
Original DQN	17.52	5.43	34.00	7.00
Rainbow-enhanced DQN	**47.54**	**17.11**	**110.00**	**17.00**

Table 2: Testing results of agents over 50 games

The Rainbow components significantly improved performance over the classic DQN and the random baseline. We considered trade-offs between computational cost and performance gains. The results validate the agent's learning capability and suggest that with extended training, further improvements are likely.

5 Conclusion

In summary, we implemented and enhanced the DQN using Rainbow components, achieving better performance than the baseline and classic DQN. Although we could not train the models long enough to reach the highest scores reported in the original paper due to computational limitations, the positive trends in the learning curves indicate that extended training would lead to continued performance improvements. Our contributions demonstrate the effectiveness of integrating advanced components and provide insights into the practical challenges of implementing RL algorithms. Future work includes extending training durations to allow the models to develop more sophisticated strategies, exploring additional DRL algorithms, and further optimizing hyperparameters.

References

1. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). *Human-level control through deep reinforcement learning.* Nature, 518, 529–533. https://doi.org/10.1038/nature14236

2. Hessel, M., Modayil, J., van Hasselt, H., et al. (2018). *Rainbow: Combining Improvements in Deep Reinforcement Learning.* Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). arXiv:1710.02298

Appendix

Source Code

The source code for our implementations is publicly available at dqn_paper_atari on the author's GitHub.