Reimplementation and Comparative Analysis of Deep Q-Network Algorithms for Atari 2600 Games

Federico Rullo
University of Bologna
federico.rullo@studio.unibo.it

Abstract

This project presents a reimplementation of the seminal work "Playing Atari with Deep Reinforcement Learning" [2]. The aim was to replicate their results on a selection of Atari 2600 games using the Deep Q-Network (DQN) algorithm. Additionally, the analysis was extended by comparing the performance of the original DQN algorithm with three variants: Double DQN [1], Dueling DQN, and Dueling Double DQN [3]. This comprehensive comparison sheds light on the effectiveness of these algorithms in tackling the challenging problem of playing Atari games using Deep Reinforcement Learning techniques.

1 Introduction

Deep Reinforcement Learning (DRL) has witnessed remarkable progress since the introduction of DQN, demonstrating the power of combining deep neural networks with Q-learning to achieve human or superhuman performance on a suite of Atari 2600 games. This work aims to reproduce and extend their findings by reimplementing the DQN algorithms and evaluating their performance against other variants.

2 Background

2.1 DQN (Deep Q-Network)

The foundation of this project is the Deep Q-Network, introduced in the paper titled "Playing Atari with Deep Reinforcement Learning" by Mnih et al. [2]. DQN amalgamates the principles of Q-learning with deep neural networks, producing remarkable results across a broad spectrum of environments, including but not limited to the Atari 2600.

Q-learning is a model-free reinforcement learning technique that tries to learn an action-value function known as a Q-function. The primary goal is to optimize the behaviour of an agent within its given environment. At the core of DQN lies its training process.

The critical aspects of DQN include:

- Experience Replay: DQN employs an experience replay buffer, which stores the past experiences (state, action, reward, next state) of the agent. During training, random minibatches are sampled from this buffer to break the temporal correlation in the data.
- Loss Function: DQN tries to minimize a loss function that quantifies the difference between predicted and target Q-values, calculated using the Bellman equation.
- Exploration vs Exploitation: DQN typically employs epsilon-greedy exploration, where with a certain probability, the agent selects a random action to explore the environment and, with probability 1-epsilon, chooses an action with the highest estimated Q-value based on the current policy.

2.2 DoubleDQN

Double DQN is an extension of the original Deep Q-network designed to mitigate a common problem known as the overestimation bias of Q-values. This bias arises when the Q-network, during its learning process, consistently overestimates the expected rewards of taking specific actions, which can lead to suboptimal or even unstable learning. Double DQN was introduced in the paper "Deep Reinforcement Learning with Double Q-learning" by Hasselt et al. [1] and addresses this issue while accomplishing it through a clever modification of the Q-learning update. The primary innovation in DoubleDQN lies in decoupling the action selection process from the evaluation of action values.

- Online and Target Networks: Double DQN employs two neural networks, the online and target networks. The online network is responsible for selecting actions based on their estimated Q-values. However, instead of using the online network to evaluate the Q-values of these actions, DoubleDQN uses the target network.
- Action Selection: The online network determines which action to take during action selection by selecting the action with the highest Q-value estimate.
- Action Evaluation: The critical difference is evaluating the selected action's value. Instead of relying solely on the online network for this evaluation, DoubleDQN uses the target network to estimate the value of the selected action. Action value estimation is achieved by applying the online network to select the action and then using the target network to estimate the associated Q-value.

The rationale behind this approach is that the overestimation bias often affects both the online and target networks similarly. By separately selecting actions and evaluating their values, DoubleDQN mitigates the risk of compounding overestimation, leading to more accurate Q-values.

The training process remains similar to that of DQN. It still uses experience replay and target networks to stabilize and expedite learning. The essential modification is in how Q-values are updated during training.

2.3 Dueling DQN & DuelingDoubleDQN

Dueling DQN & Dueling Double DQN are extensions of traditional DQN and Double DQN architectures designed to enhance the efficiency and effectiveness of Q-value estimation in Deep RL. The critical innovation of Dueling Architectures lies in the separation of Q-values into two distinct components: the Value Function (VF) and the advantage function (A(s, a)).

In the "Dueling Architectures for Deep Reinforcement Learning" by Wang et al. [3] paper, the authors recognized that not all states require the same level of granularity. Some states may benefit from modelling their intrinsic desirability (value), while others may require modelling the advantages of taking specific actions in those states. Such separation allows for more efficient learning and better generalization.

Key Features of DuelingDQN:

- Network Architecture: Dueling DQN utilizes a neural network architecture with two parallel streams: one for estimating the value function (V(s)) and the other for estimating the advantage function (A(s, a)). These streams share convolutional layers before diverging into separate, fully connected layers.
- Value and Advantage Aggregation: The outputs of the value and advantage streams are combined to ensure their combination retains the exact Q-value predictions as traditional DQN. This combination maintains the same scale and representational power as traditional Q-values while allowing the network to learn the value of a state independently of its advantage.
- Q-value Calculation: The Q-value for a specific action in a given state is calculated by combining the estimated value function and the advantage function while subtracting the mean advantage across all actions.

Dueling Double DQN builds upon these principles and incorporates a Double DQN strategy to reduce overestimation bias in Q-learning. Similar to DoubleDQN, it employs two separate Q-networks to decouple action selection from value estimation. It retains the value and advantage streams where the value function represents the intrinsic desirability of a state, and the advantage is the additional expected rewards of taking specific actions in that state.

3 Methodology

3.1 Experimental Setup

In our experiments, we select a subset of Atari 2600 games: Breakout, SpaceInvaders and Seaquest, as used in the original DQN paper [2]. However, instead of using OpenAI's Gym framework, we opt for Farama's Gymnasium framework, which offers a more stable and updated environment for evaluation. The TensorFlow library was used to build the Neural networks, and training was performed on a CPU with a Ryzen 5800x processor and 16 GB of RAM. We used the wandb library to track observations, which offers a clean and interactive environment to observe the graphs.

3.2 Reimplementation Setup

This reimplementation of the DQN algorithm closely follows the architectural and training procedures detailed in the original paper by Mnih et al.(2013) [2]. A convolutional neural network was employed to process raw pixel inputs, followed by fully connected layers to estimate Q-values for each available action. The Huber loss function was employed to ensure stability and mitigate issues related to outliers. Also, the Adam optimization algorithm was used for model updates.

Critical components of this setup include:

- Experience Replay: Experience replay was used to enhance training stability and reduce the impact of correlations in sequential data. This technique allows storing and sampling, in the form of a buffer or queue, experiences from the agent's interaction with the Atari environments, making past experiences important in training.
- Target Network: Following the DoubleDQN algorithm paper by Van Hasselt et al. (2015)[1], a target network was used to stabilize training. This network allows the evaluation of the selected actions with weights updated periodically to reduce the risk of divergence.
- Preprocessing: Following the original paper, the Atari frames were preprocessed, resizing them from a native resolution of 210x160 to 84x84 pixels. This was done thanks to the "AtariPreprocessing" wrapper provided by Gymnasium. Also, the "FrameStack" wrapper was used to stack four consecutive frames, creating an input tensor of shape 84x84x4. This stacked representation captures valuable temporal information in dynamic environments.
- Environment Selection: Experiments were conducted on a selection of the Atari game environments used by the original paper. In particular, the games used are SpaceInvaders, Seaquest and Breakout. In each of these environments, the score achieved in each episode was considered a reward signal. The "RecordEpisodeStatistics" wrapper was implemented to facilitate tracking of the scores, providing information about each episode's length and final score.
- Performance Metric: The average score per episode was calculated to enable comparison between the different algorithms. This metric made it possible to observe the performance of each algorithm over time.

3.3 Hyperparameters

Due to limited computational resources, our training is constrained to 300 episodes, unlike the 10,000 episodes specified in the original paper. This will not give the best performance for each algorithm but still makes it possible to observe and compare the different algorithms' behaviour. We also introduce an additional stopping criterion where training terminates when an acceptable average reward value is reached to expedite training.

We implement two neural network architectures: one following the template from the original DQN paper and another with a modification used for the Dueling networks. In the modified architecture, the network's last layer is split into two distinct layers: an advantage layer and a value layer, enhancing the model's representational capacity.

We summarize our hyperparameters in Table 1.

Table 1: Q-value algorithms Hyperparameters

Hyperparameter	Value
Episodes	300
Average Reward Limit	400
Batch Size	32
Learning Rate	0.001
Discount (γ)	0.99
Probability Random Action (ϵ)	1.0
Random Action Decay	0.99
Minimum Random Action Probability	0.1
Target Network Update Frequency	50 (frames)

4 Results

This reimplementation demonstrates that DQN achieves acceptable performance on the selected Atari environments, even with a significantly reduced number of training episodes compared to the original paper. Notably, DQN outperforms other algorithms in Sequest, while DuelingDQN and DoubleDQN exhibit higher scores and performance in SpaceInvaders and Breakout, respectively. However, it is worth noting that the DuelingDoubleDQN algorithm lagged behind its counterparts in the carried experiments. Further investigation is needed to ascertain the factors contributing to this waker performance. Potential causes may lie in the limited number of episodes or poor choice of parameter setting.

An in-depth analysis of the rewards obtained throughout the training was performed to understand better the performance exhibited by the DQN variants. It was possible to see varying degrees of performance across all variants. In some games, they displayed convergence to higher rewards, while others exhibited more gradual learning curves. Regarding one of the problems of DQN, which is overestimation bias, it could be observed that DoubleDQN mitigates this problem. This resulted in a more stable and accurate action value estimation over time. The consequence of this reduction was a more robust learning process and improved overall performance in games like Breakout and SpaceInvaders. On the other hand, the performance of Dueling variants showed mixed results. While there was a significant performance in games like SpaceInvaders, there were multiple performance drops, especially in the case of DuelingDoubleDQN. This suggests that the effectiveness of such a technique depends on the specific game characteristics.

We present our maximum reward obtained results in Table 2.

Table 2: Maximum Reward Obtained

Algorithms	Breakout	SpaceInvaders	Seaquest
DQN	8	600	**520**
DoubleDQN	**9**	700	360
DuelingDQN	8	**740**	320
DuelingDoubleDQN	6	655	380

5 Discussion

This reimplementation of the DQN algorithm in the context of Atari 2600 games reaffirms the algorithm's robustness and effectiveness. In particular, DoubleDQN presents itself as an enhancement over the original DQN, offering more reliable Q-value estimations. However, DQN continues to outperform DoubleDQN in certain games, suggesting that the choice of algorithm may be game-dependent.

The Dueling architecture introduced an element of variability in performance, indicating a sensitivity to the specific characteristics of each game. However, the limited resources and training time prevented exploration of whether the reduced number of training episodes exacerbated these limitations. This limitation also affected the ability to replicate the exact results presented in the original paper by Mnih et al. (2013) [2].

Despite these constraints, this experiment's results provide valuable insights into the behaviour of different DQN algorithm variations and align with the observations made in the reference paper [2].

6 Conclusions

This paper presents a successful, even if contained, reimplementation of the "Playing Atari with Deep Reinforcement Learning" paper, extending the analysis to include both DoubleDNQ and Dueling DQN variants. The obtained results emphasize the significance of algorithmic improvements in achieving state-of-the-art performance in deep reinforcement learning tasks. Additionally, they underscore the continued effectiveness of the DQN algorithm in the domain of deep reinforcement learning.

Future research efforts should overcome the limitations encountered in this experiment by extending the number of training episodes and refining hyperparameters for DuelingDQN to ensure consistent performance across diverse game environments. Finally, exploring on-policy methods like Proximal Policy Optimization, extending this comparison to have a more comprehensive view of the potential that deep reinforcement learning offers.

References

[1] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: `1509.06461 [cs.LG]`.

[2] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: `1312.5602 [cs.LG]`.

[3] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2016. arXiv: `1511.06581 [cs.LG]`.

7 Appendix A: Average Rewards Graphs

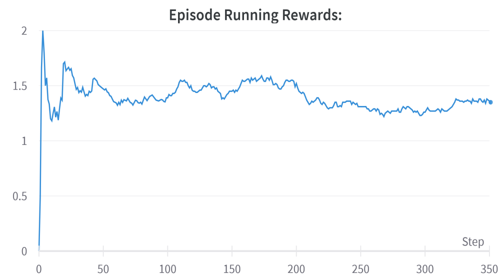

Figure 1: DQN Breakout Average Reward

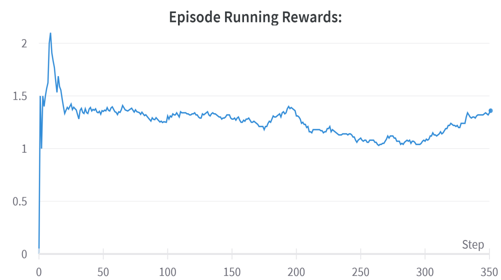

Figure 2: DoubleDQN Breakout Average Reward

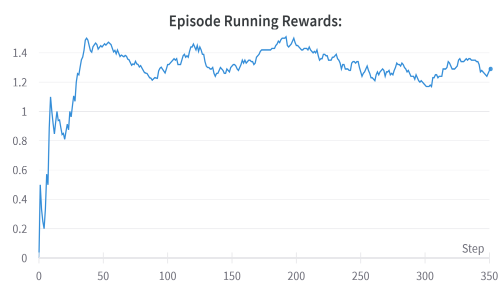

Figure 3: DuelingDQN Breakout Average Reward

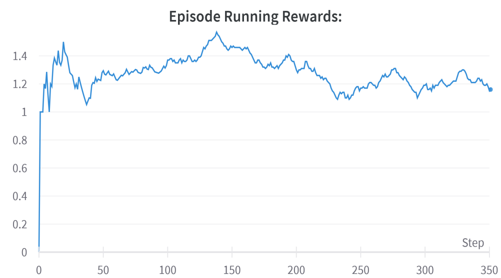

Figure 4: DuelingDoubleDQN Breakout Average Reward

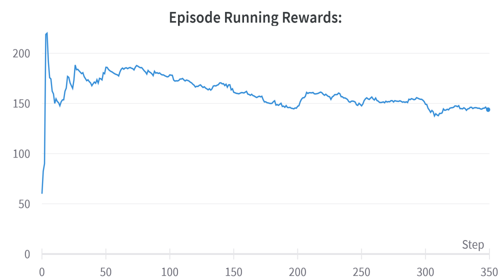

Figure 5: DQN SpaceInvaders Average Reward

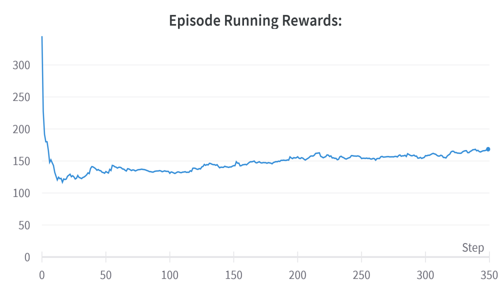

Figure 6: DoubleDQN SpaceInvaders Average Reward

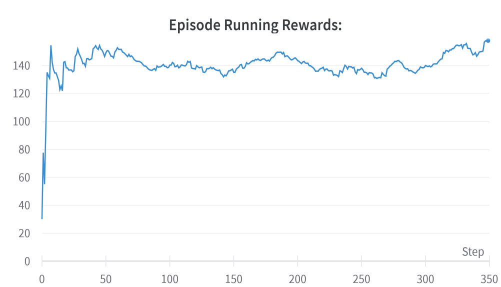

Figure 7: DuelingDQN SpaceInvaders Average Reward

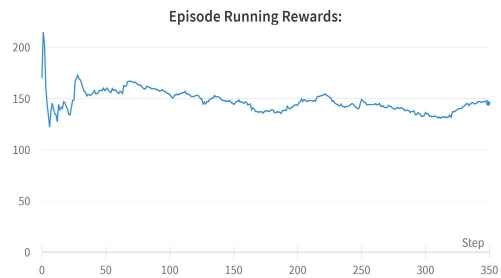

Figure 8: DuelingDoubleDQN SpaceInvaders Average Reward

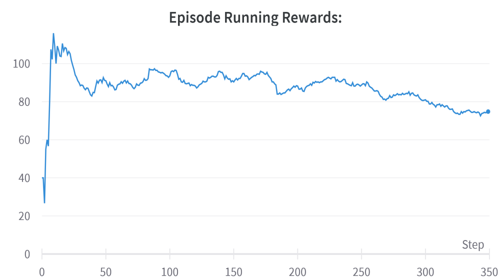

Figure 9: DQN Breakout Seaquest Reward

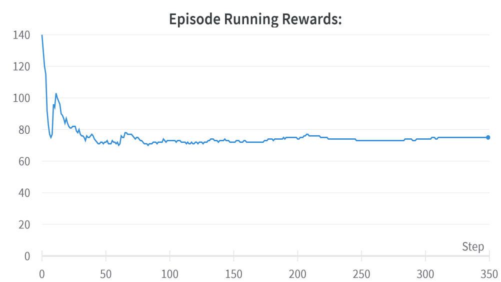

Figure 10: DoubleDQN Seaquest Average Reward

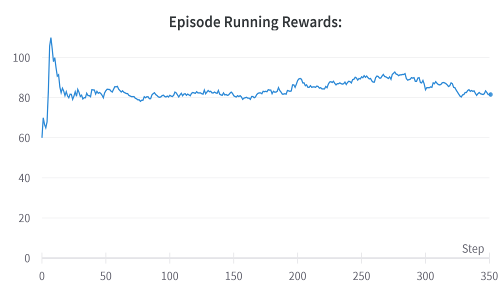

Figure 11: DuelingDQN Seaquest Average Reward

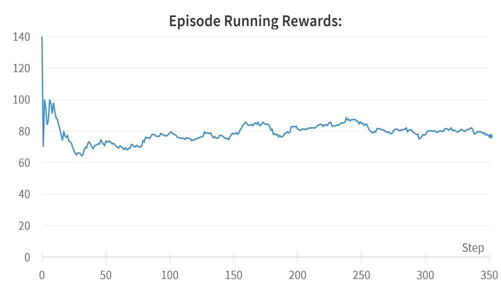

Figure 12: DuelingDoubleDQN Seaquest Average Reward