Technical report: Comparative analysis of four DQN Algorithm implementations in the Mountain Car environment

Inasse Tyouss - 201902018
Master in Big Data Technologies and Advanced Analytics
Universidad Pontificia Comillas

15/05/2024
Machine Learning II +OC

Abstract

This report presents a comparative analysis of four distinct implementations of the Deep Q-Network (DQN) algorithm applied to the Mountain Car environment, a a problem in reinforcement learning. The objective is to evaluate the effectiveness of each approach and determine which modifications yield the most significant improvements in performance. These implementations included basic DQN, DQN with performance enhancements, Dueling DQN, and DQN with prioritized replay. I'm going to discuss the structure of each algorithm, the training process, and the observed results, highlighting strengths and weaknesses of each approach.

1 Introduction

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative rewards. When there are billions of different states and hundreds of distinct actions, the Q-table becomes enormous, and it is not feasible to use a simple Q-learning model. To address this, Deep Q-Network (DQN) algorithm has to be invented. It combines the Q-learning algorithm with deep neural networks, excellent ways to approximate non-linear functions. Therefore, this algorithm uses a neural network to approximate the Q-function, thus avoiding the use of a table to represent it. The main neural network, represented by the parameters θ, is used to estimate the Q-values of the current state s and action a: $Q(s, a; \theta)$. The second, the target neural network, parameterized by θ', has the same architecture as the main network but is used to approximate the Q-values of the next state s' and the next action a'. Learning occurs in the main network and not in the target network. The target network is frozen (its parameters are not changed) for several iterations (usually around 10,000), and then the parameters of the main network are copied to the target network, thus transferring the learning from one to the other, making the estimates calculated by the target network more accurate. However, due to resource limitations on my local machine and CPUs, extensive training in my models here is not possible.

As mentioned previously, the implementations are:

- Basic DQN (*01_Initial_DQN_Implementation*)
- DQN with Performance Enhancements (*02_DQN_Performance_Enhancements*)
- Dueling DQN (*03_Dueling_DQN_Implementation*)

- DQN with Prioritized Replay (*04_Prioritized_Replay_DQN*)
- Final DQN Analysis and Visualization (*05_Final_DQN_Analysis_and_Visualization*)

2 Mountain Car environment overview

The Mountain Car environment, part of the Classic Control environments in Gymnasium, presents a quite challenging task for reinforcement learning algorithms, as it involves a car positioned at the bottom of a sinusoidal valley, where the primary goal is to apply strategic accelerations to move the car to the top of the right hill.

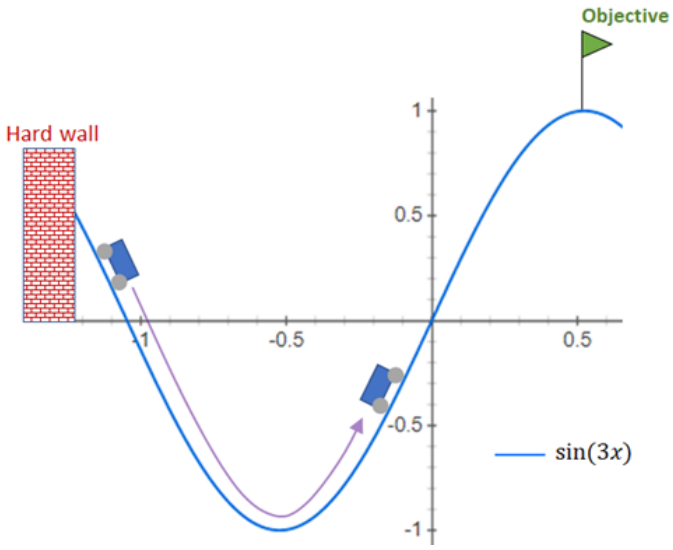

Figure 1: Mountain Car environment

2.1 Action space

The action space in this environment is discrete and consists of three possible actions:

- 0: Accelerate to the left.
- 1: Do not accelerate.
- 2: Accelerate to the right.

2.2 Observation space

The observation space is a two-dimensional array comprising:

- Position of the car along the x-axis, which ranges from -1.2 to 0.6 meters.
- Velocity of the car, ranging from -0.07 to 0.07 meters per second.

2.3 Transition dynamics

$$velocity_{t+1} = velocity_t + (action - 1) * force - \cos(3 * position_t) * gravity$$
$$position_{t+1} = position_t + velocity_{t+1}$$

Where the constants are:

- force = 0.001
- gravity = 0.0025

Collisions at either end of the position range result in the velocity being set to zero; both position and velocity are clipped to their respective ranges.

2.4 Reward structure

The agent receives a reward of -1 for each timestep until it reaches the goal, and the negative reward structure encourages the agent to find the quickest path to the goal.

2.5 Starting state

The car's initial position is uniformly randomized within the range of [-0.6, -0.4], and the initial velocity is always set to zero.

2.6 Episode termination

An episode can terminate in one of two ways:

- The car's position reaches or exceeds 0.5, indicating that the goal has been achieved.
- The episode length exceeds 200 timesteps, marking the maximum allowable steps per episode.

3 Implementation details

3.1 1. Basic DQN

In the 01_Initial_DQN_Implementation, I used a neural network with two hidden layers, each consisting of 64 neurons.

Objective: The primary aim here was to establish a reference point so I could measure the impact of enhancements, in order to help me identify the key areas that needed improvement, particularly in terms of learning stability and convergence speed.

3.2 2. DQN with performance enhancements

For the 02_DQN_Performance_Enhancements, I increased the complexity of the network by enlarging the hidden layers to 128 neurons each, raised the batch size to 128 and extended the training duration to 150 episodes, in order to provide the model with greater learning capacity and improve its stability during training.

Objective: The goal was to see if a more complex network and larger batch sizes would enable the agent to capture more intricate patterns in the data, thereby improving performance. I could see some improvements visually, but it never reached its goal.

3.3 3. Dueling DQN

In the 03_Dueling_DQN_Implementation, I implemented a dueling network architecture, which separates the estimation of the state value and the advantage of each action. This design could provide more accurate value estimations by decomposing the Q-value into value and advantage components.

Objective: By employing the dueling network, I tried to make the agent distinguish between the value of being in a particular state and the advantage of taking a specific action. This, theoretically,

should lead to better policy decisions and faster learning- however, the improvements were not as significant as expected and never reached the goal visually, but also performed worse, as the car never went up the hill and moved very slowly.

3.4 4. DQN with prioritized replay

The 04_Prioritized_Replay_DQN introduces a prioritized replay buffer, to prioritize experiences with higher temporal difference (TD) errors, allowing the agent to focus on learning from the most informative transitions.

Objective: This model should address the inefficiencies of uniform sampling in experience replay by ensuring that the agent spends more time learning from experiences that have a significant impact on its performance. This method showed marked improvements, as it allowed the agent to converge more quickly and efficiently compared to the previous approaches.

3.5 5. Detailed Analysis of Final Notebook Implementation

The 05_Final_DQN_Analysis_and_Visualization notebook utilizes identical code to the 04_Prioritized_Replay_DQN notebook as I believe I reached a pretty good model, but I've incorporated visualizations of training metrics and included a video of the best episode.

Model architecture: The agent is implemented using a Deep Q-Network (DQN) architecture that consists of a neural network with two hidden layers, each containing 64 neurons. The input to the network is the state of the environment, represented by the car's position and velocity, and the output is the Q-values for each possible action.

Prioritized experience replay: To enhance learning efficiency, the model incorporates a prioritized replay buffer, which prioritizes experiences with higher temporal difference (TD) errors so that the agent learns from the most informative transitions.

Target network: The DQN uses a target network to stabilize learning and the target network's weights are periodically updated to match those of the main Q-network, reducing the oscillations and divergence.

Training process: The agent was trained over 500 episodes to maximize cumulative rewards. Key components of the training loop include:

- **Action selection:** Actions are chosen using an epsilon-greedy strategy, balancing exploration and exploitation. The epsilon value, or the likelihood of choosing a random action, decays over time from 1.0 to a minimum of 0.01.
- **Environment interaction:** The agent interacts with the environment, receiving a reward and a new state based on its chosen action.
- **Experience storage:** Each experience (state, action, reward, next state, done) is stored in the replay buffer.
- **Mini-batch updates:** Periodically, mini-batches of experiences are sampled from the replay buffer to update the Q-network. The loss between the predicted Q-values and the target Q-values is minimized using the Mean Squared Error (MSE) loss function.
- **Reward function:** The reward function penalizes the agent with a reward of -1 for each timestep until it reaches the goal. The reward structure emphasizes the importance of reaching the goal state as quickly as possible, driving the agent to develop an efficient policy.

Interpretation of performance metrics:

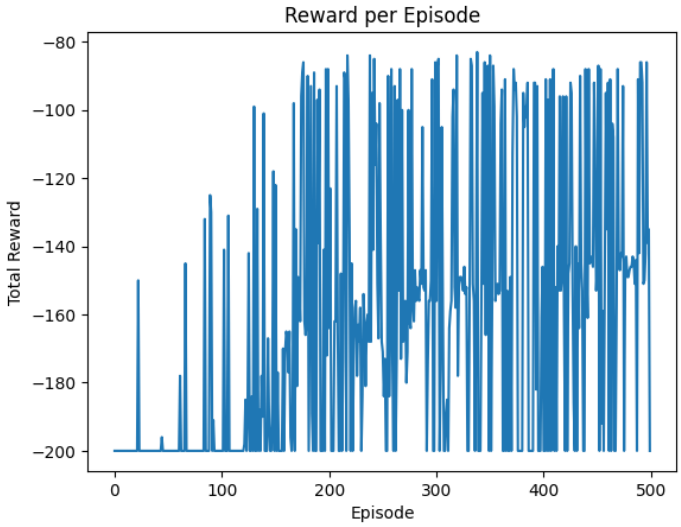

Figure 2: Plot of rewards per episode

The rewards per episode plot, as shown in the graph, illustrates the agent's cumulative rewards over the 500 training episodes. Initially, the agent consistently scores around -200, and around episode 22, we notice an improvement, achieving a reward of -150. Despite the variability and the rewards fluctuating significantly in between some episodes as the model shows both improvement and regression, occasional peaks demonstrate that the agent has a lot of potential to discover effective strategies.

Examples of best episodes:

- Episode 84: Total reward: -132.0, Epsilon: 0.00998645168764533
- Episode 89: Total reward: -125.0, Epsilon: 0.00998645168764533
- Episode 90: Total reward: -130.0, Epsilon: 0.00998645168764533
- Episode 102: Total reward: -141.0, Epsilon: 0.00998645168764533
- Episode 106: Total reward: -131.0, Epsilon: 0.00998645168764533
- Episode 125: Total reward: -142.0, Epsilon: 0.00998645168764533
- Episode 130: Total reward: -99.0, Epsilon: 0.00998645168764533
- Episode 133: Total reward: -129.0, Epsilon: 0.00998645168764533
- Episode 139: Total reward: -101.0, Epsilon: 0.00998645168764533
- Episode 167: Total reward: -98.0, Epsilon: 0.00998645168764533

Exploration vs. exploitation trade-off: The epsilon-greedy strategy balances exploration and exploitation. High exploration (high epsilon) ensures diverse experiences, while later stages focus on exploitation (low epsilon) to refine the policy. The agent's balance between exploration and exploitation certainly could need some further tuning.

4 Conclusion and discussion:

Overall, while the model showed significant fluctuations, indicating some instability in the learning process, it frequently achieved the objective of reaching the hill. As training progressed over more episodes, the agent's performance improved, demonstrating its ability to accomplish the task more reliably, and visually, the agent often managed to reach the goal quickly.

Reflecting on this environment, despite thinking at first that it was a simple setup, it presents a considerable challenge due to the necessity for directed exploration. If the agent never encounters the reward, it cannot learn an effective policy and random exploration alone is insufficient, as the car requires specific momentum to reach the goal.

It is then important to incentivize the agent to explore new parts of the environment, with an effective strategy, a "bonus-based exploration," which involves rewarding the agent for visiting states it has not encountered frequently, encouraging the agent to explore more thoroughly. This approach is particularly powerful for learning in environments with sparse rewards like Mountain Car.

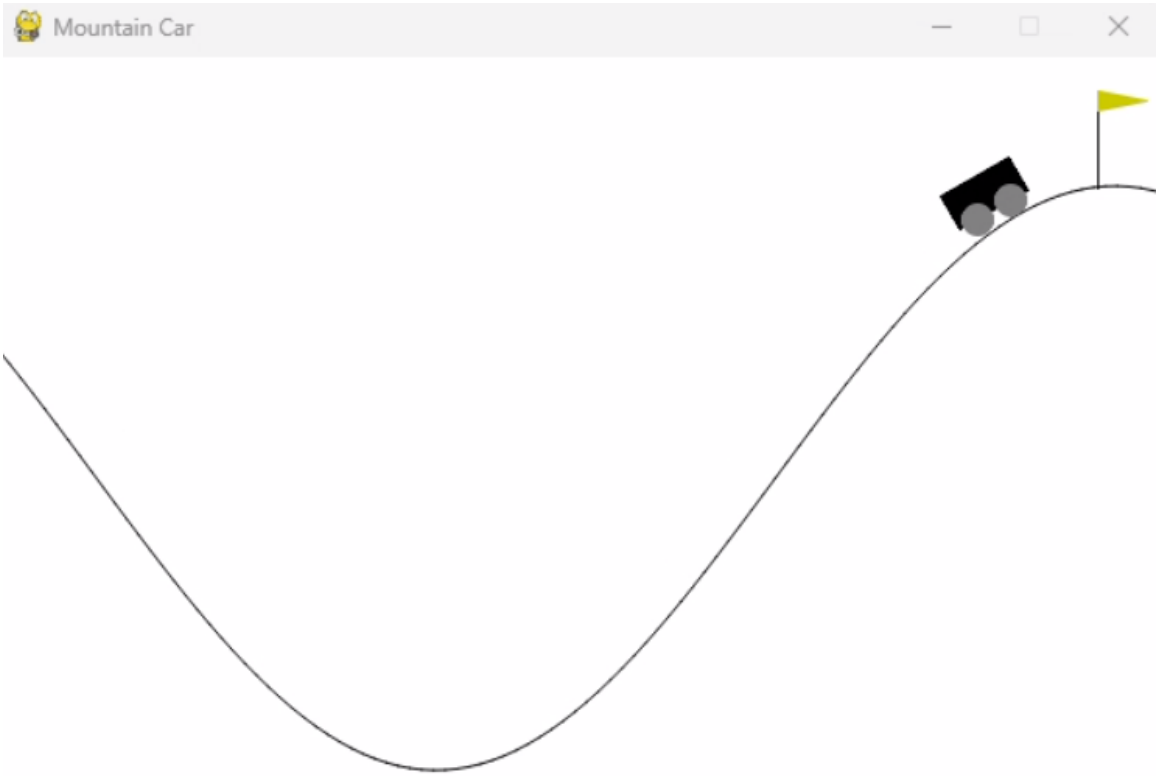

Figure 3: Car reaching the objective