Comparison between the DQN and DuelingDQN architectures

Andrea Pinto
andrea.pinto2@studio.unibo.it

January 21, 2023

Abstract

In the following project have been studied the differences in the performances of the Deep-Q-Network and the Dueling Deep-Q-network architectures on the 'Lunar Lander' environment provided by the Gymnasium API.

1 Environment

In this section will be provided a brief description of the environment.

1.1 State and actions

The Lunar Lander in a classic rocket trajectory optimization problem. In this project has been used the environment that has a discrete action set. The action space is composed of four discrete actions:

- 0: Do nothing
- 1: Fire left orientation engine
- 2: Fire main engine
- 3: Fire right orientation engine

The state is an 8-dimensional vector, containing the coordinates of the lander, its linear velocities, its angle, its angular velocity, and two booleans that represent whether each leg is in contact with the ground or not.

1.2 Rewards

The aim of the lander is to land on the landing pad (which is always at coordinates (0, 0)). The environment provide many reward but they can be summarized in:

- the lander gets a negative reward if the engines are firing

- the lander has a positive reward if it's able to move slowly toward the ground maintaining a horizontal angle
- the reward is positive if the legs of the lander are on the ground
- the episode receives a massive positive reward if the lander is able to land safely, a negative one in case it crashes

An episode is considered 'solved' if the agent is able to score 200 points.

1.3 Episode

An episode finishes when:

- The lander crashes (its body gets in contact with the ground)
- The lander gets outside of the viewport (x coordinate > 1)
- The lander is not awake (it does not move nor collide with any other body)

2 Agent

In this section will be described the setting of the problem and the characteristic of the agent. What will be described here will be valid for both the DQN and the Dueling DQN.

2.1 Setting

At each time step of the environment the agent will be asked to select an action $a \in \mathcal{A}$ given the actual state $s \in \mathcal{S}$, the environment will return the next state s_{t+1} and the reward r_{t+1} obtained as a consequence of the action chosen, together with the information of whether the next state is a terminal one or not. The goal of the agent is to select actions that maximize the future discounted return, defined as $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$ where T is the terminal state. It's possible to define the optimal action-value function as the maximum expected return achievable by following any policy in a state s and taking some action a, $Q^*(s,a) = max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$. From the Bellman equation, it's possible to deduce that if the optimal value $Q^*(s,a')$ of the next time step s' was known for each possible action a', then the optimal strategy would be to select the action maximizing the expected value of $r + \gamma Q^*(s',a')$. For this reason, the basic idea is to use the Bellman equation as an iterative update in order to approximate the action-value function, in our case with deep neural networks, and make it converge to the optimal one. The loss used to train the network will be

$$L_i(\theta) = \mathbb{E}\Big[\Big((r + \gamma max_{a'} Q(s', a'; \theta' | s, a)) - Q(s, a; \theta)\Big)^2\Big]$$

Where θ represent the main network and θ' is the target network that is updated every few step with the weights of the main network. In the experiments, the Huber loss (also known as smooth L1 loss) has been tried and it yielded better results compared to the mean squared error.

The agent has been trained with the same hyperparameter, the only thing that changes are the architectures used to make decisions.

Epsilon-greedy policy In the beginning the deep neural network won't be able to know the exact/good action values. For this reason, it's been used the ϵ-greedy policy which is a good way to balance exploration and exploitation since the action will be selected with a greedy strategy with probability 1 - ϵ and a random action will be selected with probability ϵ. The ϵ will decay exponentially over the episodes, this means that at the beginning the agent will explore a bit the environment and learn to approximate good action value estimates and then with a certain probability it will exploit its knowledge to make decisions.

Experience replay The experience replay is a technique that consists in saving the 'experiences' of the agent at each time step. An experience is composed of a state, the action taken in that state, the reward obtained from the environment, and the next state that follows. In addition, it is stored in the experience also a boolean value called 'done' that indicates if the next state is a terminal state. The experience can be represented as the following tuple

$$e_t = (s_t, a_t, r_t, s_{t+1}, done)$$

During training will be sampled a batch of experience from the experience replay buffer. This technique is very useful: learning by consecutive samples is inefficient because the samples have a strong correlation, with this method it's possible to generalize better by reducing the variance of the updates and it also allows for greater data efficiency since each experience could be used potentially in many weight updates.

2.2 DQN

The architecture of the Deep-Q-Network used in this project is different from the original one. Since in the Lunar Lander environment the representation of the state is an 8-dimensional vector, the convolutional feature extractor has been removed. The network takes in input the 8-dimensional state vector, has an input layer with 64 nodes, a hidden layer with 64 nodes, and an output layer with 5 outputs, which correspond to the possible actions.

2.3 Dueling DQN

The Dueling DQN is conceptually very similar to the DQN. It takes in input the state and estimates the Q values for all the possible actions in that state. The main difference is that in the network there are two different streams that learn how to approximate the value function $V(s)$ and the advantage function $A(s, a)$, defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, which are then combined in an aggregation layer to compute $Q(s, a)$. The idea is that the Dueling architecture is able to learn which states are more valuable without having to learn the effect of each action in each state, this is useful in states where actions do no affect the environment in a relevant way. The advantage function can be seen as a relative measure of the importance of each action. In the implementation the convolutional feature extractor has been removed, the network takes in input the state vector and uses only a fully-connected layer to compute the two streams. The aggregation layer is defined as follows:

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

Double DQN algorithm The Dueling DQN architecture uses the Double DQN algorithm which aims at reducing the overestimations of the Q values. This is done by modifying the target as follow

$$y_t = r + \gamma Q(s', argmax_a Q(s', a; \theta); \theta')$$

In standard Q-learning the max operator is used to both select and evaluate an action while by decoupling selection and evaluation, it's possible to reduce the overoptimistic value estimates.

3 Results

As a reference for the evaluation of the result will be considered the average score over 100 episodes.

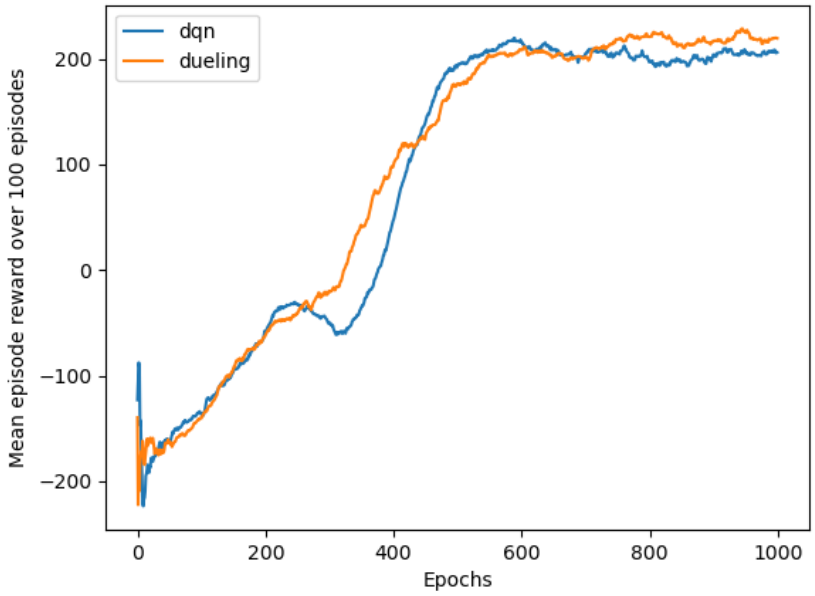

Figure 1: Comparison of the DQN and DuelingDQN performance

From the graph it's possible to notice that the two networks behave very similarly. This is probably due to the fact that the Lunar Lander has short episodes, with a small state space and action space. For these reasons it's not possible to appreciate the qualities of the Dueling DQN. It's possible to observe though that the Dueling architecture, even if it's composed by a larger amount of parameters, it's able to learn as fast as the standard DQN. Furthermore, while the performances of the DQN oscillate near the score of 200 (needed to 'solve' the environment), the Dueling architecture continues to accumulate experience and gain an average score that increases over time.