A Comparison Between 5 DQN Variants on LunarLander-v2 Environment

Vincenzo Maria Stanzione
Department of Computer Engineering (DISI)
University of Bologna, Italy
vincenzo.stanzione@studio.unibo.it

Abstract

In the last few years, we assisted to a flourishing season of improvements for Deep Q-Network (DQN) algorithms and architectures; this has led to the creation of a great number of variants in respect to the first model proposed [1]. In this work, we will experiment 5 different variants of DQN, in particular: DQN, DQN with Fixed Q-Targets (an improved version of DQN that exploits a target network to overcome convergence problems), Double DQN (DDQN), Dueling architecture for DQN and finally we will combine these approaches into Dueling Double DQN (D3QN). We will create intelligent agents to successfully master Open AI gym LundarLander-v2 discrete environment and we will compare the performance obtained.

1 Introduction

Reinforcement Learning aims at learning how to take optimal decisions in unknown environments by trying to maximize the obtained reward through time. Agents have to update their own beliefs about the world, about which actions are good and which are not. Temporal difference and off-policy learning are the foundations of this learning behaviour [2]. Algorithms based on Q-learning, such as DQN, are driving Deep Reinforcement Learning research towards solving more and more complex problems and achieving super-human performance in several domains [3].

Even though Q-learning (and then DQN) have led to great performance in domains such Atari 2600 games [4], DQN has been proven to suffer from overstimation, a phenomenon that can deeply impact the convergence speed. In recent years several efforts have been made to overcome this problem, with the proposal of several variants that we will discuss in the next paragraph.

2 Selected DQN versions

In this section, we will briefly recap the DQN versions we will experiment. We will start by presenting the DQN base algorithm and then we will point out the differences brought by the successive versions. For each version, we will consider to exploit a simple Experience Replay, even if is possible to implement more sophisticated approaches, such as Episodic Memory [5].

2.1 DQN

For clarity purposes, we report in Algorithm 1 the DQN "classic" algorithm, exposed in [1], to better understand the modifications brought by other variants.

Algorithm 1: DQN with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function network Q with random weights θ
foreach *episode* **do**
  Initialize s_0 initial state
  foreach *step of episode* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    if s_{j+1} *is terminal state* **then**
      $y_j = r_j$;
    else
      $y_j = r_j + \gamma \max_a Q(s_{j+1}, a; \theta)$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
  end foreach
end foreach

2.2 DQN with Fixed Q-Targets

As shown in Algorithm 2, one of the first improvements to the initial DQN architecture has been to use in addition to the main neural networks (NN), a twin target NN, updated every C steps (we copy the weights of the main model into the target model). The target network is used to compute action-value functions for future states. This allows the network to converge more smoothly, especially in the early training phases when the training is very unstable [4].

Algorithm 2: DQN with Fixed Q-Targets and Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function network Q with random weights θ
θ^- initial target network Q^- parameters, θ^- copy of θ
foreach *episode* **do**
  Initialize s_0 initial state
  foreach *step of episode* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    if s_{j+1} *is terminal state* **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma \max_a Q^-(s_{j+1}, a; \theta^-)$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    Every C steps set $\theta^- = \theta$
  end foreach
end foreach

2.3 Double DQN

Another improvement was proposed by van Hasselt et al. [6]: Double DQN (DDQN). DDQN tries to solve an overestimation bias of Q-Learning by decoupling selection and evaluation of the actions, using similarly to the previous case two twin network: one to select the action that maximize future action-value function and one to compute the actual value of it. The algorithm is shown in Algorithm 3.

Algorithm 3: DDQN with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function network Q with random weights θ
θ^- initial target network Q^- parameters, θ^- copy of θ
foreach *episode* **do**
  Initialize s_0 initial state
  foreach *step of episode* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Define $a'(s_{t+1}; \theta) = argmax_a Q(s_{j+1}, a; \theta)$
    if s_{j+1} *is terminal state* **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma Q^-(s_{j+1}, a'(s_{j+1}; \theta); \theta^-)$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    Every C steps set $\theta^- = \theta$
  end foreach
end foreach

2.4 Dueling DQN

Dueling architecture explicitly separates the representation of state-values $V(s)$ and (state-dependent) action advantages $A(s, a)$ from $Q(s, a)$. The dueling architecture consists of two streams that represent the value and advantage functions, as shown in Figure 1. The two streams are combined via a special aggregating layer to produce an estimate of the state-action value function [7].

The gist of the dueling architecture is that we can learn which states are (or are not) valuable, without having to learn the effect of each action for each state. This is particularly useful in states where actions do not affect the environment in any relevant way. This architecture consent to have a more stable learning process for the network. In addition, we learn state-value functions for each experience we do, while for classic DDQN we update action-value functions only for one selected action a in a state s.

We can apply this type of architecture to DQN to obtain Dueling DQN and to Double DQN to obtain Dueling Double DQN (D3QN).

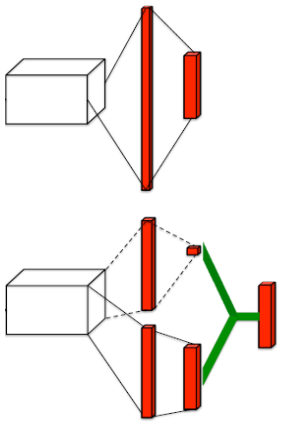


Figure 1: Classic architecture vs Dueling Architecture [7]

3 Environment

As stated in the abstract, we will use different intelligent agents based on DQN variants previously exposed to master LunarLander-v2 discrete environment. The objective of LunarLander-v2 is to safely land in the landing pad of coordinates (0,0). We have 4 discrete actions available: do nothing, fire left engine, fire main engine, fire right engine. An episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10 points. Firing the main engine is -0.3 points each frame. Fuel is infinite, so an agent can learn to fly and then land on its first attempt.

The environment is considered solved if we achieve an average reward of 200 for over 100 consecutive episodes. Then, our aim is to reach at least 200 average reward over 100 consecutive episodes for the agents proposed. To do this, we will use as input the state of the environment composed of a tuple of 8 elements[1].

4 Testing the agents

For this purpose, a neural network composed of 3 hidden layers has been used (Figure 2). In particular:

- Input layer accepting inputs of shape (1,8)
- Fully connected layer of 128 neurons
- Fully connected layer of 64 neurons
- Fully connected layer of 32 neurons
- Output of the network, a fully connected layer of 4 neurons

For the dueling architecture (Figure 3), the skeleton of the network is the same. However, between the last but one layer and the output layer, a layer has been added to compute the state-value function and the advantages (then aggregated in another specific layer).

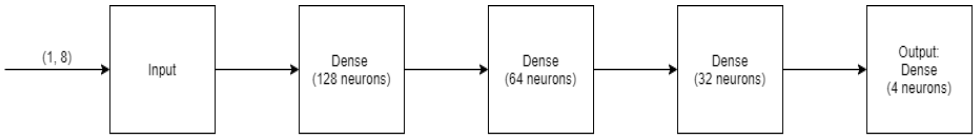


Figure 2: Classic NN

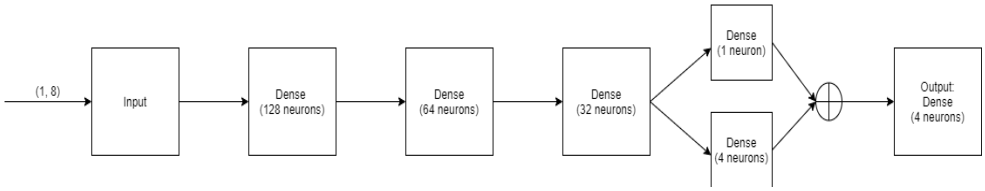


Figure 3: Dueling NN

The hyperparameters used for all the training are reported in Table 1. Regarding the soft update option, it was tested only for DDQN due to computational constraints and the fact that the performance were worse than the hard update[2].

[1] Refer to `https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py`

[2] If we perform a "soft update", the target network weights will be updated as $\theta^- = \theta^-(1-\tau) + \theta\tau$. We refer to "hard update" when we update the weights of the target network as $\theta^- = \theta$, so with $\tau = 1$

Table 1: Hyperparameters, "*" means used for DDQN and DQN + target network

Optimizer	Adam
Learning Rate	0.0005
Experience Replay Size	100000
Minimum Experience Replay Size	128
Episodes	500
Discount Rate (γ)	0.99
Epsilon max	1
Epsilon min	0.01
Epsilon decay	0.9995
Batch Size	64
Tau (τ)*	0.1
Update Frequency (C)*	100

5 Results and comparison

After training all the agents for 500 episodes, with training time of 6-7 hours each, all the agents reached > 200 average reward, computed incrementally over the last 100 episodes occurred, with the exception of DDQN with soft update option. The best agent is without a doubt D3QN for the score obtained and the stability during the training phases in general (Figure 4). This is evident from Figure 5, that represents the reward average considering all 500 episodes. For D3QN it is still > 200 at the end, even if the early noisy phases of training are considered. Furthermore, it is important to say we reached a score peak of 321.56 for D3QN and the problem was solved after 136 episodes, which currently achieves a 4th place in the LunarLander-v2 leaderboard.

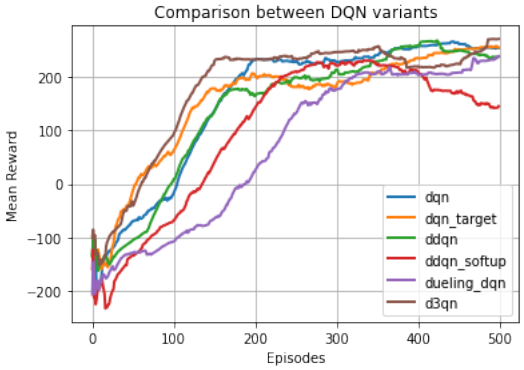


Figure 4: Average Reward computed incrementally over the last 100 episodes occurred

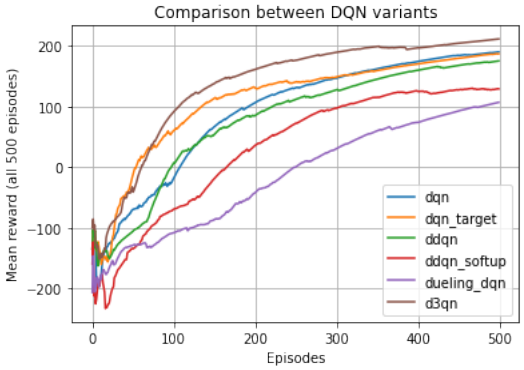


Figure 5: Average Reward over all 500 episodes

References

[1] V. Mnih et al., “Playing Atari with Deep Reinforcement Learning,” Dec. 2013, Accessed: Jul. 04, 2021. [Online].

[2] R. S. Sutton and A. G. Barto, "Reinforcement Learning. A Bradford Book", 2018.

[3] A. Cini, C. D’Eramo, J. Peters, and C. Alippi, “Deep Reinforcement Learning with Weighted Q-Learning,” ResearchGate, Mar. 2020, Accessed: Jul. 05, 2021. [Online].

[4] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[5] Z. Lin, T. Zhao, G. Yang, and L. Zhang, “Episodic Memory Deep Q-Networks,” Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), Accessed: Jul. 05, 2021. [Online].

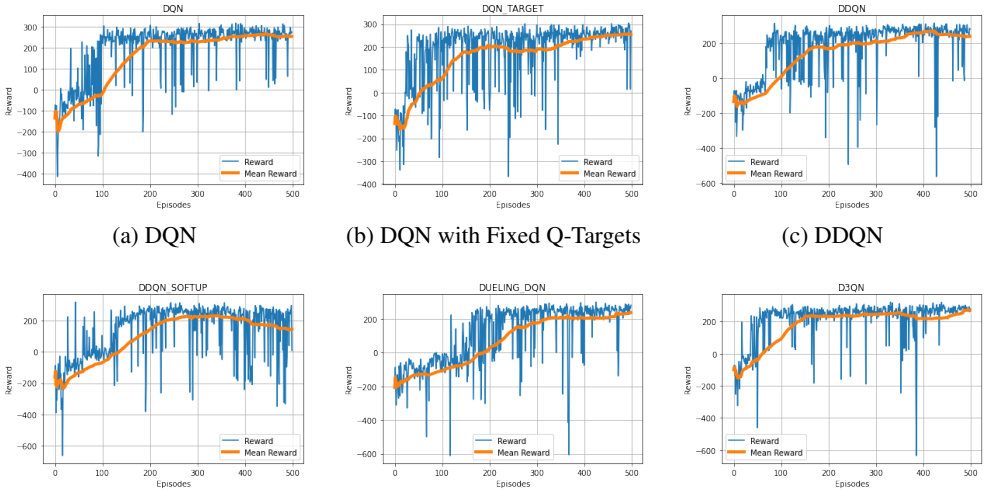
[6] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning,” Dec. 2015, Accessed: Jul. 05, 2021. [Online].

[7] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” Apr. 2016, Accessed: Jul. 05, 2021. [Online].

Appendix

The weights of the models in .h5 format and the reward, mean reward and steps history are available at this link, as well as a Jupyter Notebook containing the code in TensorFlow+Keras for the agents and the training.

Furthermore, for completeness, in Figure 6 the reward and average reward history for each agent are reported.



(a) DQN (b) DQN with Fixed Q-Targets (c) DDQN

(d) DDQN with Soft Update (e) Dueling DQN (f) D3QN

Figure 6: Reward and Average Reward over the last 100 episodes