Playing retro games with Dueling DQN

Michele Morisco (505252)
Department of Computer Science, University of Pisa, Pisa, Italy
Intelligent System for Pattern Recognition, Academic Year 2021/2022

Abstract

The following document illustrates the work done for the Intelligent System for Pattern Recognition course project. The aim of the project is to implement and experiment with a Dueling DQN to play some retro games using OpenAI Gym environments.

1 Introduction

Teaching skills, tactics, and strategies to a computer program, to play a video game, is a far more challenging task. In recent years, extensive research was carried out in the field of reinforcement learning and several algorithms were introduced, aiming to learn how to perform human tasks such as playing video games.
This paper is about a Dueling Deep Q-Network implemented to experiment with its performance in different video game environments. During the experiments, three games are tested to compare the results between distinct approaches.
The report is divided into four parts: Section 2 defines the background of Deep Reinforcement Learning applied for this project; Section 3 describes the video game environments and the methodology of models used for the experiments; in Section 4, we will see the experimental results achieved. Lastly, Section 5 gives a brief conclusion that describes the experience.

2 Background

An agent interacts with an environment ϵ, and the emulators used in the experiments have a sequence of actions, observations, and rewards. At each time step, the agent selects an action a_t from the set of legal game actions, $A = \{1, ..., K\}$.
The action is passed to the emulator and modifies its internal state and the game score. The agent observes an image $x_t \in R^d$ from the emulator, which is a vector of raw pixel values representing the current screen. In addition, it receives a reward r_t representing the change in in-game score.
Since the agent only observes images of the current screen, the task is partially observed. Many emulator states are impossible to fully understand the current situation from only the current screen x_t.
We have a large and finite Markov decision process (MDP) in which each sequence is a distinct state. As a result, we can apply standard reinforcement learning methods for MDPs by using the complete sequence s_t as the state representation at time t. The agent's

goal is to interact with emulators by selecting actions that maximize future rewards. Our future discounted return at time t as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. Defining value function is described by the basic form of the Bellman equation:

$$v(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s) v(s')$$

it describes the iterative relationship between states.
We use the Action-Value function to represent the return obtained after executing the π strategy in the s state.

$$q_\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a]$$

where π is a policy mapping sequences to actions. It can also be decomposed into

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

Now, finding the optimal strategy can be equivalent to solving the optimal Action-Value function:

$$q_*(s, a) = \max_a q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s', a')$$

We can use Q-learning to solve the problem. Similar to Value Iteration, Q-learning updates the Q value as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

This algorithm is model-free: it solves the reinforcement learning task directly using samples from the emulator ϵ, without explicitly constructing an estimate of ϵ.
It is also off-policy: it learns about the greedy strategy $a = \max_a Q(s, a)$ while following a behavior distribution that ensures adequate exploration of the state space.

3 Data and Methodology

3.1 Data description

OpenAI Gym is an open-source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments.
In this project, we have three different environments *Pong*, *Boxing*, and *Super Mario Bros.*. The first two games were simulated via Stella and the Arcade Learning Environment (ALE)[Machado et al. (2017)] which contains a set of Atari 2600 environments.

3.1.1 *Pong*

The player controls the right paddle and competes against the left paddle controlled by the computer. The reward is the difference between the agent's score and the opponent's

Figure 1: Screenshots from three environments used for the experiments: (*Left-to-right*) Pong, Boxing, Super Mario Bros.

score points.

3.1.2 *Boxing*

The player (in white) fights an opponent (controlled by the computer, it is black) in a boxing ring and scores points for hitting the opponent. If the player scores 100 points, the opponent is knocked out. The reward is the same as *Pong*.

3.1.3 *Super Mario Bros.*

The environment is simulated using the nes-py emulator [Kauten (2018)] and its reward function assumes the objective of the game is to move as far right as possible, as fast as possible, without the player dying.

$$R = v + c + d$$

where v is the difference in the agent's position values between states, c is the difference in the game clock between frames and d is a death penalty that penalizes the agent for dying in a state. Last but not least, in this project we consider two types of *Super Mario Bros.* environments to restrict the action space: right only and simple movement. In the first one, the agent goes only right direction; on the contrary, in the second one, the agent can go both left and right. The additional information about the environments used is shown in Table 1.

3.2 Methodology

3.2.1 DQN

A Deep Q-Network (DQN) approximates a state-value function in a Q-Learning framework with a neural network. In that use case, they take in several frames of the game as

Environment	Action space	Observation space
Pong	6	(210, 160, 3)
Boxing	18	(210, 160, 3)
SMB (RIGHT ONLY)	5	(240, 256, 3)
SMB (SIMPLE MOVEMENT)	7	(240, 256, 3)

Table 1: Some additional information about the environments.

an input and output state values for each action as an output. It is usually used in conjunction with **experience replay**, for storing the episode steps in memory for off-policy learning, where samples are drawn from the replay memory at random. Additionally, the Q-Network is usually optimized towards a frozen target network that is periodically updated with the latest weights every k step.
The Alg.1 shows the DQN with an experience replay. [Mnih et al. (2013)]

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode=1, M **do**
  Initialise sequence $s_1 = \{x_1\}$
  and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for t=1, T **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image $x_t + 1$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

3.2.2 Dueling DQN

Dueling DQN [Wang et al. (2015)] considers dividing the Q network into two parts. The first part is only related to the state S and has nothing to do with the specific action A. This part is called the value function, denoted as $V(S, w, \alpha)$, the second part is related to both the state S and the action A. This part is called the Advantage Function, denoted as $A(S, A, w, \beta)$, then finally the value function can be replace with

$$V(S; w, \alpha) + (A(S, A; w, \beta) - \frac{1}{A} \sum_{a' \in |A|} A(a, a'; w, \beta))$$

w is the public parameter, α is the private parameter of V, and β is the private parameter of A.
Applying a "naive" summing is not possible because, given the function Q, we cannot determine the values of V and A since that is unidentifiable so applying the mean helps with this issue, it does not change the relative rank of the A (and hence Q) values, preserving any greedy or ϵ-greedy policy based on Q values from the equation.

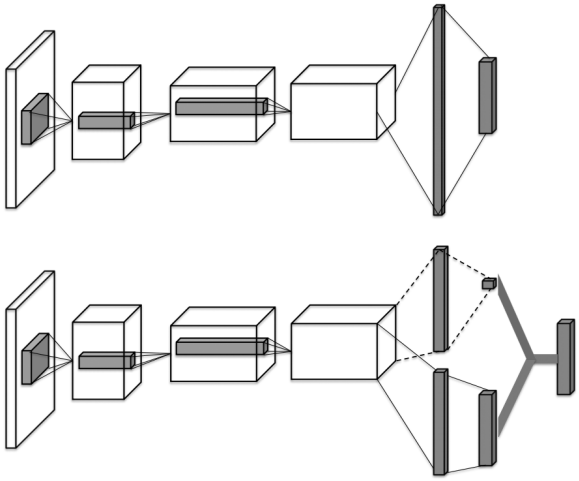

Figure 2: A single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action.

3.2.3 Double Dueling DQN

It is similar to Dueling DQN but in this case, the model was changed to use two Q networks, one strategy network is used to update the network and select actions, and the other target network is used to calculate the target Q value. The target network is updated with a delay. In the Alg. 2 we can see the pseudo-code of a double DQN. [Mnih et al. (2015)]

4 Results

In this section, we will see the results achieved from the experiments. For each environment, they are tested by three models introduced in the previous section.
Generally, they use the same hyperparameters between models.

4.1 *Pong*

In *Pong* environment, it used the following hyperparameters shown in Table 2 and we can notice the three models used the same parameters except for Double Dueling DQN that which used less memory capacity and a Huber loss function, being more complex with respect to other models, the memory capacity was reduced due to tackle the out-of-memory

Algorithm 2 Double Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\widehat{Q}$ with weights $\theta^- = \theta$
for episode=1, M **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for t=1, T **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = argmax_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image $x_t + 1$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C states resets $\widehat{Q} = Q$
  end for
end for

error imposed by limited resources.

Hyperparameter	DQN	Dueling DQN	Double Dueling DQN
Initial epsilon	1	1	1
Final epsilon	0.01	0.01	0.01
Epsilon decay	30k	30k	30k
Number of frames	2M	2M	2M
Optimizer	Adam	Adam	Adam
Memory capacity	100k	100k	50k
Loss function	MSE	MSE	Huber
Learning rate	1e-5	1e-5	1e-5
Batch size	32	32	32
Gamma	0.99	0.99	0.99

Table 2: Hyperparameters used in the various models for Pong environment

Fig. 3 shows the average reward per episode achieved in the training phase.
We notice the plots are similar between them but we can see a difference in the curves of reward: in the DQN model, the curve grows slightly slower than Dueling DQN model that it is better than DQN most of the time, indicating that the training is faster. On the contrary, Double Dueling DQN seems its training is slower equal to DQN but we can see it reaches the best reward gain faster than Dueling at a certain point. But, we can notice it has more big fluctuation as DQN, on the contrary, Dueling DQN has a small fluctuation with the respect to other ones.
Lastly, all models were able to train effectively the agent to play Pong very well with not

so many episodes.

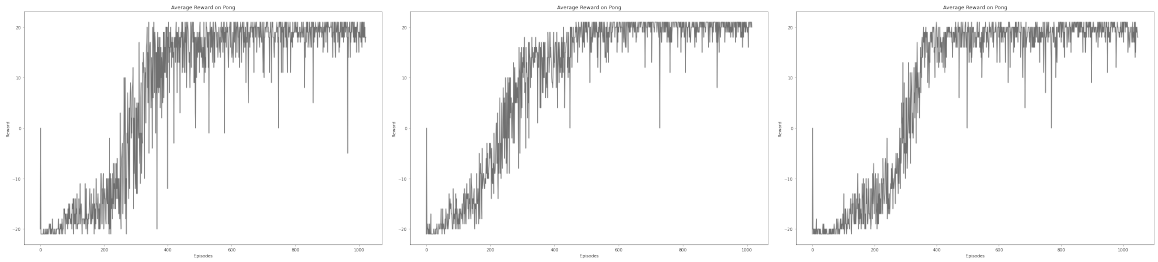

Figure 3: These plots show the average reward per episode for the Pong environment. They are the results respectively (*Left-to-right*) by DQN, Dueling DQN, and Double Dueling DQN.

4.2 *Boxing*

The *Boxing* environment is slightly more complex than *Pong* environment, indeed *Boxing* has a bigger action space and it has game rules more complex than the previous one.
It used the following hyperparameters shown in Table 3 and they are similar to the previous environment's parameters.

Hyperparameter	**DQN**	**Dueling DQN**	**Double Dueling DQN**
Initial epsilon	1	1	1
Final epsilon	0.01	0.01	0.01
Epsilon decay	30k	30k	30k
Number of frames	2M	2M	2M
Optimizer	Adam	Adam	Adam
Memory capacity	100k	100k	50k
Loss function	MSE	Huber	Huber
Learning rate	1e-5	1e-5	1e-5
Batch size	32	32	32
Gamma	0.99	0.99	0.99

Table 3: Hyperparameters used in the various models for Boxing environment

In Fig. 4 we can see all three models' training is slower than the previous environment but it is an expected behavior because *Boxing* is a more complex game with respect to *Pong*. In more depth, we notice Dueling DQN is faster than other ones. For this environment, Double DQN performs slightly better than Dueling DQN, indeed Dueling remains below the Double model for the last 200 episodes. In addition, with *Boxing* we have more fluctuation in reward with respect to *Pong*.
The three models have trained well the agent, even if it has not achieved a high reward but for the aim of the experiments, have got acceptable results. Naturally, if the agent had observed more frames he would get a higher reward.

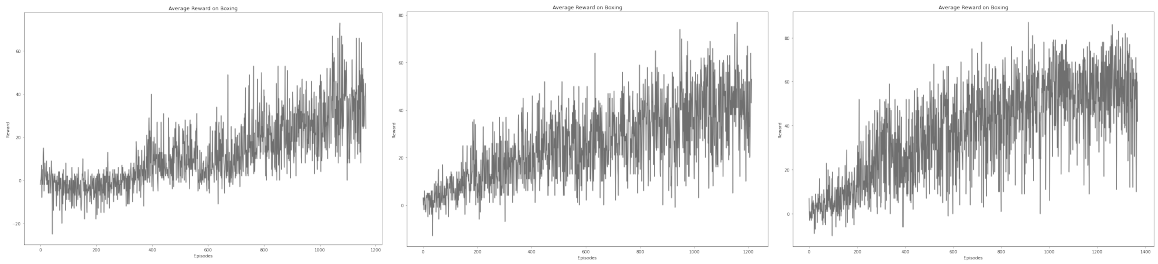

Figure 4: These plots show the average reward per episode for the Boxing environment. They are the results respectively (*Left-to-right*) by DQN, Dueling DQN, and Double Dueling DQN.

4.3 *Super Mario Bros.*

The last environment is the most complex used for the experiments, even if it has a smaller action space than *Boxing*, the game world of *Super Mario Bros.* has multiple obstacles and combines different actions.
The following hyperparameters are shown in Table 4 and we can notice a reduced number of episodes with respect to previous environments, indeed in the first two games the agent has observed many frames but for this environment, it is not reached more than 1.5M frames. This difference is due to the very long training time that prevented prolonged use of resources. In addition, the memory capacity is smaller than *Pong* and *Boxing*.

Hyperparameter	DQN	Dueling DQN	Double Dueling DQN	Simple Movement
Initial epsilon	1	1	1	1
Final epsilon	0.01	0.01	0.01	0.01
Epsilon decay	50k	50k	50k	50k
Number of episodes	7500	7500	5700	5700
Optimizer	Adam	Adam	Adam	Adam
Memory capacity	30k	30k	30k	30k
Loss function	Huber	Huber	Huber	Huber
Learning rate	2.5e-4	2.5e-4	2.5e-4	1e-4
Batch size	32	32	32	32
Gamma	0.99	0.99	0.99	0.99

Table 4: Hyperparameters used in the various models for Super Mario Bros. environment

Finally, Fig.5 shows the plots of the average reward for the *Super Mario Bros.* environment. In the first experiment, the movement of Mario is set up on 'Right Only'.
We notice more fluctuations than in other games, This behavior is present because *Super Mario Bros.* has a more complex environment as said previously, but during the training, the agent seems to be able to learn to play and as we can see, the reward grows quite well. DQN works worst than other models, on the contrary, Dueling DQN and Double Dueling DQN seem to have the same performances, but Double DQN has better results than Dueling. Indeed, during the training phase, the agent reached the end of level 1101 times against the 435 times for Dueling DQN.

As said in the previous game environment, the agent hasn't reached a high reward, as well as it appears a heavy fluctuation, indeed the training would require more and more observations to accomplish in the best way the task. But, with Dueling DQN and Double Dueling DQN, the agent performs very well, particularly with the Double one.

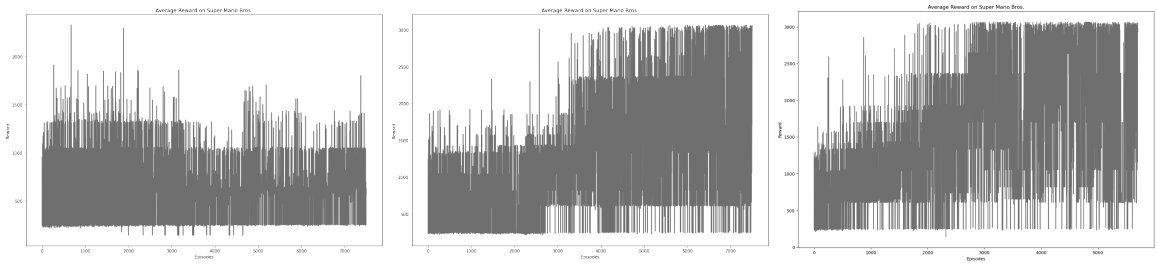

Figure 5: These plots show the average reward per episode for Super Mario Bros. environment. They are the results respectively (*Left-to-right*) by DQN, Dueling DQN, Double Dueling DQN.

In the last experiment, the best model, i.e. Double Dueling DQN, is tested with the 'Simple Movement' version. Using this environment version, the training is slower, indeed the agent has reached the end of level 883 times. But, the results achieved are always acceptable.

Environment	**Episodes**	**Average reward**	**Score/Goal**
Pong (DQN)	1022	18.7	(1 - 21)
Pong (Dueling DQN)	1016	20.6	(1 - 21)
Pong (Double D-DQN)	1048	18.3	(0 - 21)
Boxing (DQN)	1166	35.6	(100 - 73)
Boxing (Dueling DQN)	1212	40.4	(100 - 66)
Boxing (Double D-DQN)	1368	46.2	(100 - 54)
Environment	**Episodes**	**Average reward**	**Times of level completed**
SMB (DQN)	7500	660.1	0
SMB (Dueling DQN)	7500	1356.4	435
SMB (Double D-DQN)	5700	2767.7	1101
SMB (Simple Movement)	5700	1969.4	883

Table 5: A summary of results achieved from different environments. The average reward is computed from the last 10 rewards.

	DQN	D-DQN		Double D-DQN	
	Score	**Score**	**%DQN**	**Score**	**%DQN**
Pong	18.7	20.6	110.2	18.3	97.9
Boxing	35.6	40.4	113.5	46.2	129.8
Super Mario Bros.	660.1	1356.4	205.5	2767.7	419.3

Table 6: Comparison of scoring achieved from different environments.

5 Conclusions

The report has shown the implementation of Dueling DQN and demonstrated its performance to master difficult control policies for several retro game environments using OpenAI Gym.
By the results, *Pong* is the most simple environment, and as expected all agents can be able to learn to play it very well.
On the contrary, with *Boxing* and *Super Mario Bros.* the agents need more frames to observe for improving their performances. Despite that, the agents succeeded to play the two games, especially with Dueling DQN and Double Dueling DQN. The last one performs better than the Dueling one.

References

Kauten, C. (2018). *Super Mario Bros for OpenAI Gym.* GitHub. Retrieved from `https://github.com/Kautenja/gym-super-mario-bros`

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2017). *Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents.* arXiv. Retrieved from `https://arxiv.org/abs/1709.06009` doi: 10.48550/ARXIV.1709.06009

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning.* arXiv. Retrieved from `https://arxiv.org/abs/1312.5602` doi: 10.48550/ARXIV.1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015, February). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. Retrieved from `http://dx.doi.org/10.1038/nature14236`

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2015). *Dueling network architectures for deep reinforcement learning.* arXiv. Retrieved from `https://arxiv.org/abs/1511.06581` doi: 10.48550/ARXIV.1511.06581