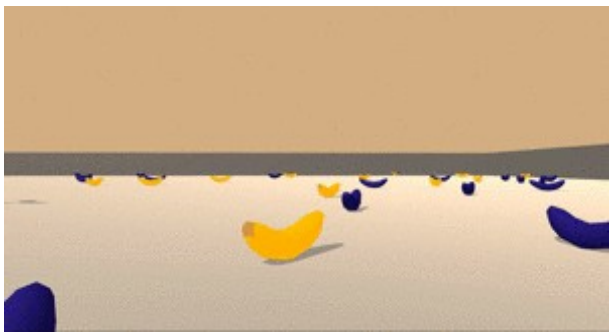
Project 1 : Navigation

Fabrice R. Noreils

Introduction

As a reminder, for this project, we train an agent to navigate (and collect bananas!) in a large, square world.



A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

- **0** - move forward.
- **1** - move backward.
- **2** - turn left.
- **3** - turn right.

The task is episodic, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

For this project, I implemented three different agents :

- One based on DQN ;
- One based on Double-DQN ;
- One based on Dueling-DQN.

In the remaining section, I will describe the different experiments I conducted along with some conclusions.

Agent based on DQN

As a reminder please find below the different steps of the DQN algorithm which is described in [1].

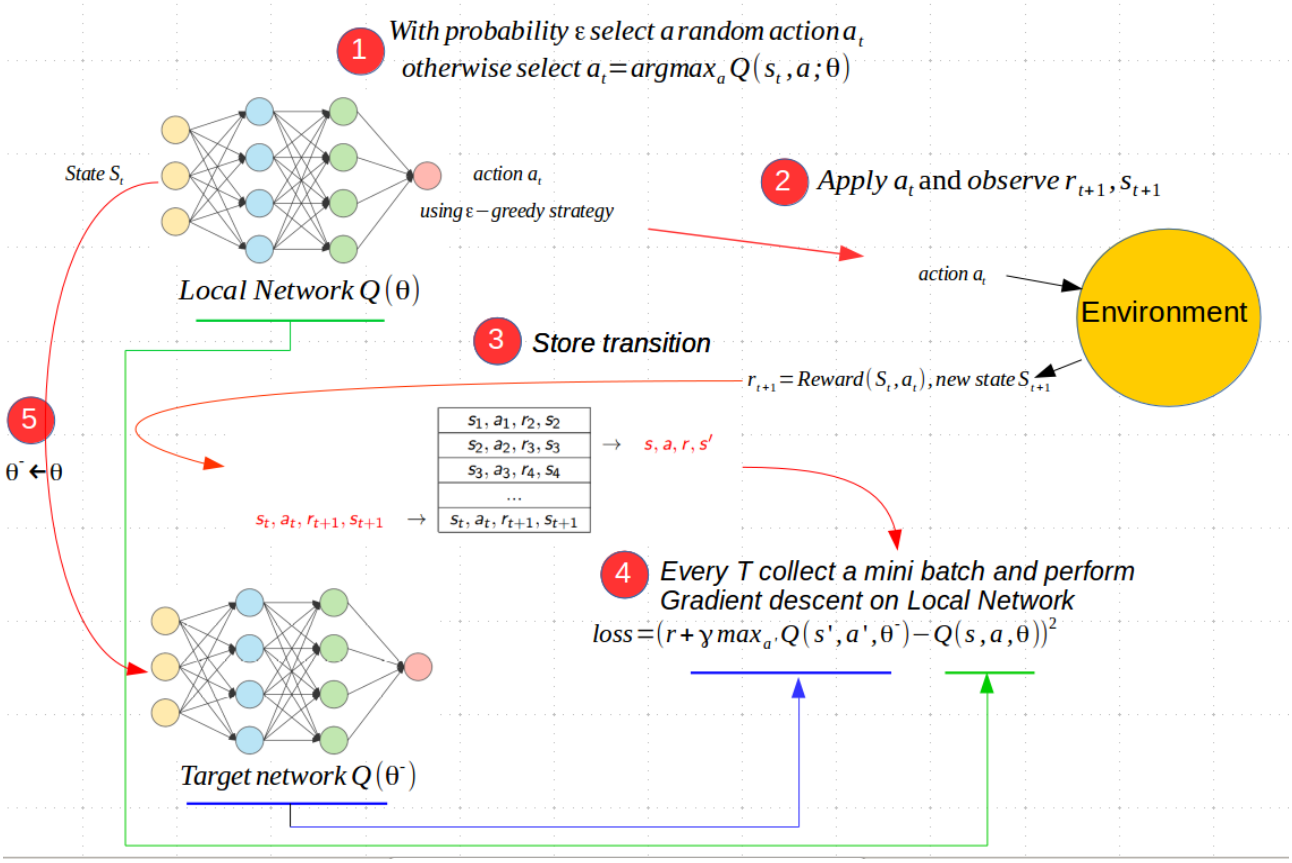


Figure 1: Different steps of the DQN Algorithm.

The table 1 below depicts the hyper parameters associated with their numerical values for the 8 tests I kept.

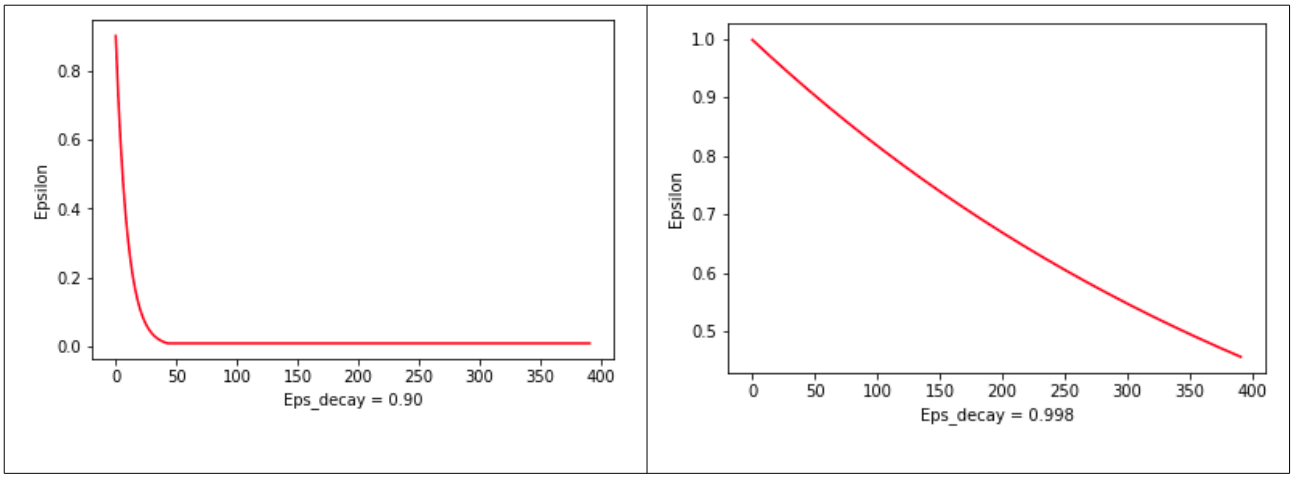
After preliminary tests, I decided to focus on two hyper-parameters, namely :

- **eps_decay** which defines the exploration/exploitation ratio ;
- **fc_units,** which defines how precise is our « curve fitting » I will say.

Parameters	Init.	Test1	Test2	Test3	Test4	Test4	Test5	Test6	Test7
LR	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4	1e-5	1e-5
Eps_decay	0.995	0.98	0.95	0.90	0.90	0.90	0.92	0.98	0.95
fc_units	128	128	128	128	196	256	512	128	128
Batch size	64	64	64	64	64	64	64	64	64
Tau	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
Gamma	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Buffer size	1e5	1e5	1e5	1e5	1e5	1e5	1e5	1e5	1e5
Episodes	577	424	398	392	**314**	411	508	859	846

Table 1: Experiment results for DQN

First it is important to notice that there is a big difference in terms of Exploration/Exploitation between eps_decay = 0.90 and eps_decay = 0.998 for 400 episodes as it is shown on the picture below :



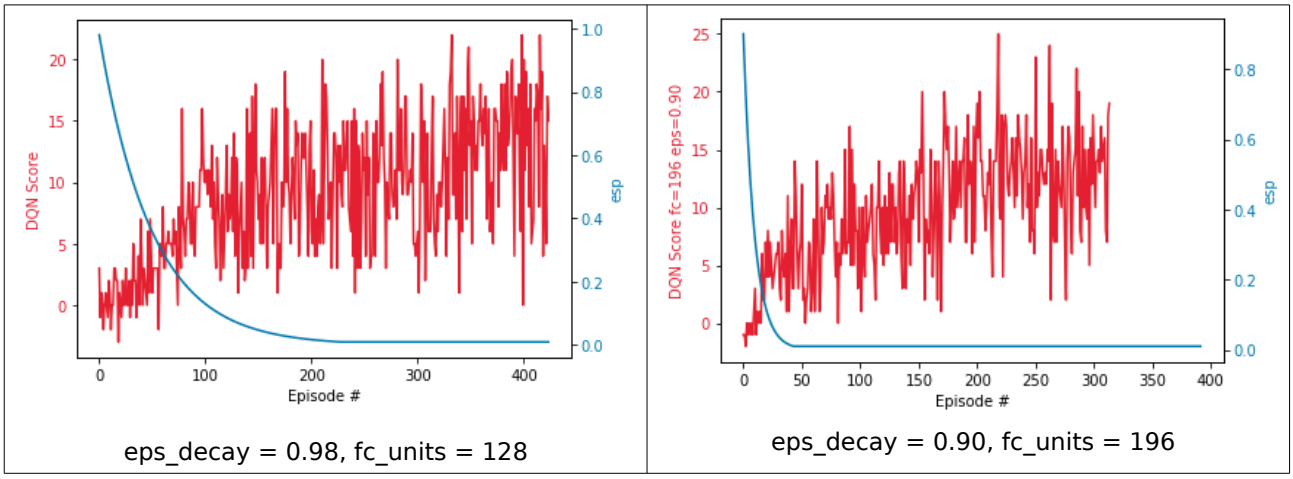
Indeed with eps_decay = 0.90, epsilon falls to 0.01 in less than 50 episodes, meaning that a random selection of an action occurs at the very beginning of the training only.

Despite this fact, I can observe that by decreasing the value of eps_decay from 0.998 to 0.90 and keeping unchanged fc_units, the agent is able to solve the game from 577 episodes to near 390 episodes. This is an interesting result because it looks like it is not necessary to explore heavily the states space.

And if I keep eps_decay = 0.90 and increase fc_units to 196, the game is solved in 314 episodes.

However if I keep increasing fc_units, the number of episodes to solve the game increase as well.

Therefore eps_decay = 0.90 and fc_units = 196 seems to be a good trade-off to solve the game in a limited number of episodes – see Table 2 for some examples of graphics displaying the score per episode.



eps_decay = 0.98, fc_units = 128

eps_decay = 0.90, fc_units = 196

Table 2: Examples of score versus number of episodes for a DQN agent.

Agent based on Double-DQN

Double -DQN has been introduced in [2] and the picture below provides a short synthesis of the addressed problem and the solution proposed by the authors.

$$Q(s,a;\theta') = r(s,a) + \gamma \max_{a'} Q(s',a',\theta')$$

Q target — Reward of taking action a at current state s — Discounted max Q value among all possible actions at next state

By calculating the TD target, we face the following problem: how are we sure that the best action for the next state is the action with the highest Q-value?

We know that the accuracy of Q values depends on what action we tried and what neighboring states we explored.

The risk lies that while learning, we may observe an extremely positive reward that is very rare or was not even necessarily because we did that action, and that action was actually a suboptimal one (instead of the true optimal action). As a result, when we choose the max over the estimates we are using a biased estimate for the optimal action; it is actually suboptimal.

The solution is:

when we compute the Q target, we use two networks to decouple the action selection from the target Q value generation.

- The local DQN network to select what is the best action to take for the next state (the action with the highest Q value).
- The target network to calculate the target Q value of taking that action at the next state.

$$Local\ Network\ Q(\theta)\ used\ to\ select\ action$$
$$Target\ Network\ Q(\theta^-)\ used\ to\ evaluate\ action$$
$$loss = (r + \gamma\, Q(s', \max_{a'} Q(S',a',\theta), \theta^-) - Q(s,a,\theta))^2$$

Based on the previous tests with DQN, I started with fc_units = 196 and eps_decay = 0.90, and DDQN needed 483 episodes to solve the game – see Table 3 for more details about the results.

Therefore I decided to set fc_units = 256 and DDQN outperformed DQN.

Parameters	Test1	Test2
LR	5e-4	5e-4
Eps_decay	0.90	0.95
fc_units	196	256
Batch size	64	64
Tau	1e-3	1e-3
Gamma	0.99	0.99
Buffer size	1e5	1e5
Episodes	483	**297**

Table 3: Experiment results for DDQN

As a conclusion I will say that it is necessary to increase the number of units in order to get better results – noticed that DQN with eps_decay = 0.90 and fc_units= 256 solved the games in 411 episodes.

See Table 4 for some examples of graphics displaying the score per episode.

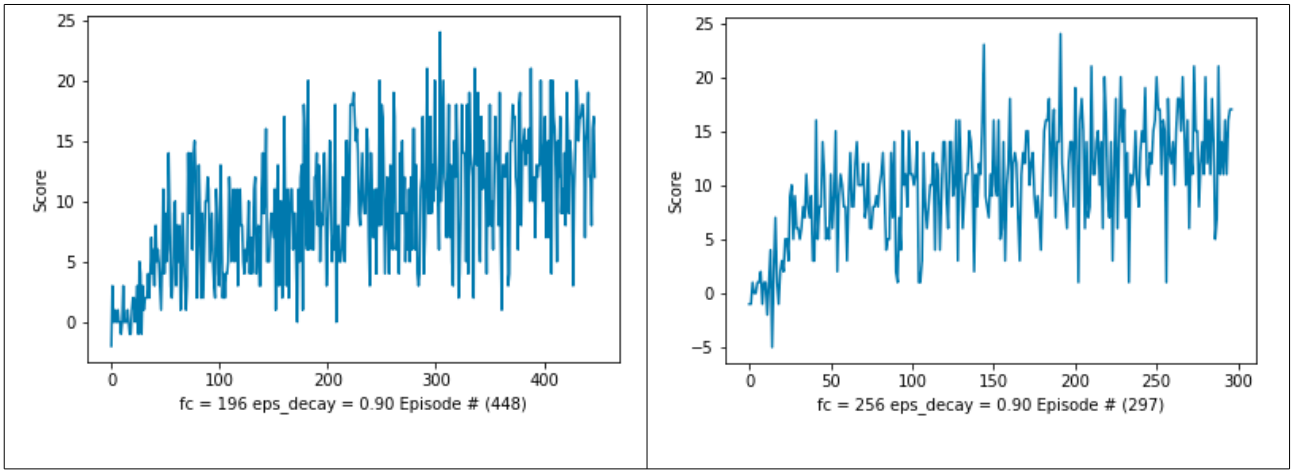


Table 4: Examples of score versus number of episodes for a DDQN agent.

Agent based on a Dueling-DQN

The main difference with respect to DQN relies on the architecture of the neural nets which has two heads :

- One to compute Advantage function ;
- One to compute Value function.

The core algorithm remains the same as in DQN.

The advantage of the dueling network over standard Q-Networks is especially prominent when the number of actions is large. For standard Q-Networks, when the variation between actions is small, the Q-Network effectively has to learn the same value for all actions while each update only modifies the Q value of one action.

Figure 2 describes the architecture of the network alongs with the parameters that will be tuned during the experiments.

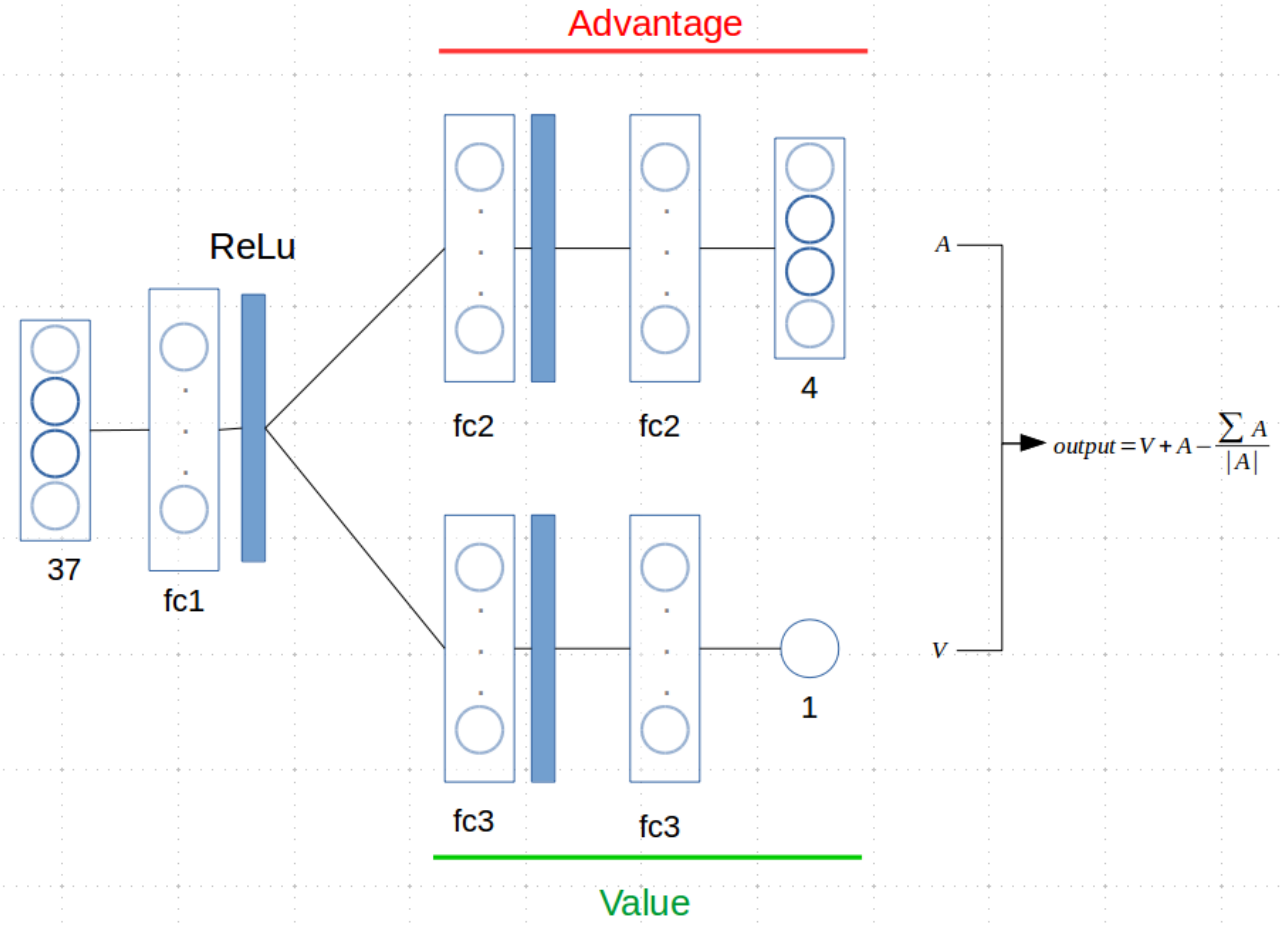


Figure 2: Architecture of the Dueling network.

Table 5 below lists all the hyper-parameters considered for Dueling-DQN. The main difference with respect to DQN and DDQN is that there are two parameters to define the number of units per group of layers: one for the Value "sub network" and one for the Advantage "sub-network".

I noticed that :

- The fc layer to compute Value must be larger than the one to compute the Advantage in order for the Dueling-network to perform well.
- As it was the case for DDQN, I had to increase the number of units for each sub-network in order to outperfom DQN

Indeed by setting eps_decay = 0.90, fc_units(Advantage) = 196 and fc_units(Value)=256, the Dueling-DQN agent was able to solve the game in 297 episodes.

Parameters	Test1	Test2	Test 3
LR	5e-4	5e-4	5e-4
Eps_decay	0.90	0.90	0.90
fc_units Advantage	128	256	196
Fc units Value	196	196	256
Batch size	64	64	64
Tau	1e-3	1e-3	1e-3
Gamma	0.99	0.99	0.99
Buffer size	1e5	1e5	1e5
Episodes	310	377	**276**

Table 5: Experiment results for Dueling-DQN

See Table 4 for some examples of graphics displaying the score per episode.

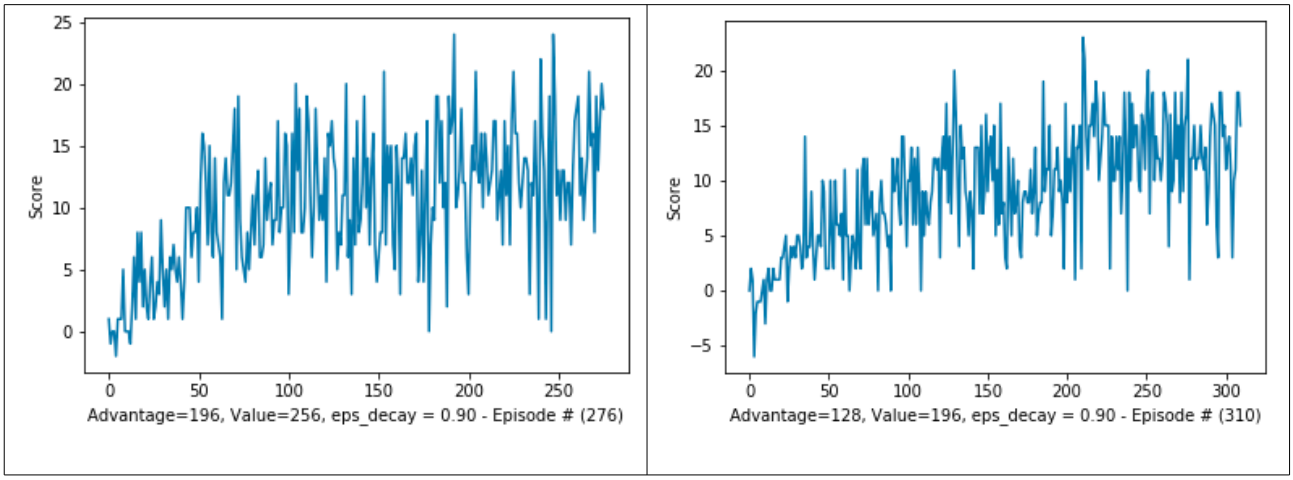


Table 6: Examples of score versus number of episodes for a Dueling DQN agent.

Conclusion

I implemented three different agents (DQN, DDQN and Dueling DQN) to solve the game in which an agent is moving while collecting yellow bananas while avoiding green bananas.

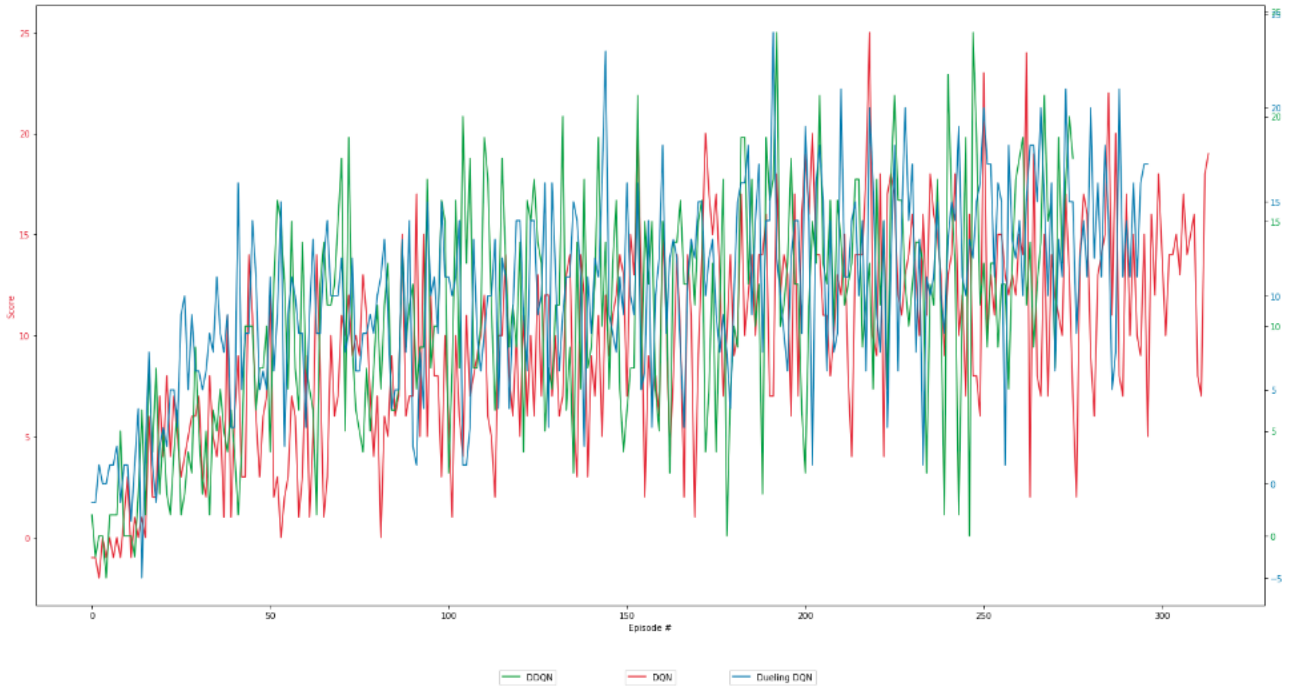
I noticed that, at least for that game, reducing eps_decay played an important role in reducing the number of episodes to solve that game. It probably means that there is no need to explore the states that much and the agent can exploit quickly the early explored states.

By tuning the different fc layers I can see that Dueling DQN outperforms DDQN which outperform DQN (see Table 7)

Algorithm	Episodes	Weights
DQN	314	checkpoint_dqn_196_2.pth
DDQN	297	checkpoint_ddqn_256.pth
Dueling-DQN	276	checkpoint_deling_dqn_3.pth

Table 7: Best performance for each agent.

The picture below is a tentative to compare the score of the three agents:



As a future work, I will implement DQN/DDQN/Dueling DQN with a prioritized buffer to see if it is possible to reduce the number of episodes to solve the game.

Bibliography

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*, 529-533.

[2] Hasselt, H.V., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. *AAAI*.

[3] Wang, Z., Freitas, N.D., & Lanctot, M. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *ICML*.