# Comparing Deep Q-Learning and Double Deep Q-Learning in Space Invaders

**Bálint Bujtor**
University of Bologna
Bologna, BO 40121
balint.bujtor@studio.unibo.it

## Abstract

In this project I compare two a popular Reinforcement Learning method, Deep Q-Learning and and improved version of it, Double Deep Q-Learning from a performance perspective. I seek the answer for the question if Double Deep Q-Learning can really decrease the loss and thus, improve the performance. As my results show Double Deep Q-Learning can indeed outperform the simple Deep Q-Learning algorithm, however I did not achieve such a significant difference between the two methods as in the original paper.

## 1 Introduction

### 1.1 Q-Learning

Q-Learning is an off policy reinforcement learning method based on temporal difference control, trying to find and execute the best action in any given state. It is off policy because the algorithm not necessarily chooses the action with the highest reward, but with a given possibility it executes a randomly chosen action. It is based on temporal difference which means it calculates the estimates based in part of other estimates without waiting for the final outcome (called bootstrapping).

Q-Learning is model-free reinforcement learning, it tries to learn a policy to maximise the total reward. It is called model-free algorithm because the learned $Q$ directly approximates the $q_*$ independently from the policy $\pi$, hence it does not need a model of the environment.

The core of the algorithm is a Bellman equation as a simple value iteration update [1], using the weighted average of the old value and the new information:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)}^{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

Figure 1: Bellman equation

Q-Learning is a tabular-method because it stores every state-action pair in a matrix or table. Even though with Q-Learning the $q_*$ is independently approximated from a given $\pi$ policy, the policy still matters since it determines the visited state-action pairs and the only requirement for Q-Learning is to keep all of the state-action pairs updated.

Storing the state-action pairs in a table can lead to success when the number of state-action pairs are low, but in many implementations it is not the case. The big number of state-action pairs can lead

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

to memory problems. To tackle this problem, Q-Learning commonly uses function approximation since it replaces the state-action table with a function approximation with a finite number of input parameters which leads to memory decrease and speed increase.

### 1.2 Deep Q-Learning

One solution to use function approximation is the usage of artificial neural networks. With artificial neural networks it is possible to approximate even nonlinear functions. However, reinforcement learning is unstable or divergent when a nonlinear function approximator is used. The reason of this is the correlation between the sequence of observation. This can be eliminated by using an experience replay or buffer, from which we randomly choose states (experience), thus eliminating the correlation between consecutive inputs. One of the first implementation of the aforementioned techniques was presented by the Google DeepMind team [2]

### 1.3 Double Deep Q-Learning

Because the future maximum approximated action value in Q-learning is evaluated using the same Q function as in current action selection policy, in noisy environments Q-Learning is known to overestimate action values under certain conditions. [3] Using Double Q-Learning reduces the observed over-estimations, as hypothesized, but that this also leads to much better performance on several games and outperforms the original DQN algorithm. In my project I would like to prove this statement presented by the Google DeepMind team.

## 2 The Environment

The two agents are going to be tested in two different environments, during two different runs. In order to be able to test the performance of the two methods I set all of the random seeds available in the algorithm to a fixed number. The two methods are going to be compared using their best trained parameters. On the final training I loaded the best parameters to all of the convolutional neural networks.

### 2.1 Space Invaders

The Space Invaders is an Atari 2600 game developed by Tomohiro Nishikado in 1978. It is considered to be one of the most influential video games of all time. During the game-play the player has to control a spaceship which shoots the invading aliens. The ship can move up, down, left and right, can shoot at the aliens and also can do nothing during a step. By shooting the aliens the player receives points and the goal is to maximise the total amount of points throughout the game. The game ends when the player loses all of its 3 lifes. The player loses a life when one of the invading aliens hits the spaceship.

I am using the OpenAI Gym environment to take care of every interaction done with the environment, input- and output-wise, thus focusing only on the algorithmic part of the problem. [4] I have chosen the *SpaceInvaders-v0* environment, where the observation inputs are images of the game.

**Observation:** an environment-specific object representing your observation of the environment. This essentially is used as our input for the function approximation. In my case the observation is an image of the game at every moment. Every image is a *210*160*3* coloured image in *uint8* format (2.1).

Table 1: Observation Space in Space Invaders

| | |
|---|---|
| Width | 160 |
| Height | 210 |
| Channel number | 3 |
| Pixel values | 0-255 |

**Actions:** The possible actions from what the agent can choose from (2.1). These actions are determine how the agent can interact in the environment. Essentially, this is the output of the function approximator. (In my case, the output of the artificial neural network.)

Table 2: Action Space in Space Invaders

| Number | Action |
|---|---|
| 0 | No operation |
| 1 | Fire |
| 2 | Move Up |
| 3 | Move Right |
| 4 | Move Left |
| 5 | Move Down |

**Reward:** amount of reward achieved by the previous action. Our goal is always to increase the total reward.

**Done:** whether it's time to reset the environment again. If *True* indicates the episode has terminated. (When the spaceship loses all of its lifes.)

## 3 The Agents

As I mentioned before I compare two different agents, in the same game and environment in order to compare their performances. One of the agents uses Deep Q-Learning, meanwhile the other agent is based on Double Deep Q-Learning. In order to be able to reproduce certain scenarios and situations the random seeds are fixed in the game. I collect the rewards and after 100 episodes the means of the rewards, to be able to evaluate the performance measures. I also save the episode durations and the means of the episode durations in the same way as the other metric. In each episode I calculate the total amount of reward received and I save the network's parameters every time it achieves a higher total reward. In case of Double DQN I also save the target network's parameters.

### 3.1 Deep Q Network

I use a convolutional neural network as the deep q network. It has 3 convolutional layers. Since I implemented frame stacking to reduce training time, the first layer has an input channel number of 4, and the output channel number of 16. The following layers have 16 input and output channel number each. All of the layers are connected by a ReLU activation layer. The last layer of the network is a fully connected layer with the output size of the number of possible actions in the game (in my case 6). The total number of trainable parameters in the network is 621558.

The Double DQN agent uses the exact same neural network as the target network.

## 4 Results

Below are the results after training both of the agents with the same fixed random seeds using the same initial model parameters for both the DQN and the DDQN agents.

In order to maximise the performance of the agents I did a hyperparameter optimisation to find the ideal hyperparameters for both agents. In order to find the best parameters I did a grid search to in multiple rounds. The purpose of doing it in multiple rounds was to save time and resources on my computer. I only checked the most promising combinations and dropped the less fruitful ones. For each combination I did 50 episodes of warm up where I filled up the replay buffer and then did 75 episodes of training. My comparison metric was the average rewards of each game after the last played episode.

First, I did a grid search to find the best batch size - convolution size - kernel size combination (4). Afterwards, with the best batch size - convolution size - kernel size combination, I searched for the

Table 3: Grid search for hyperparameters

| Batch size | Convolution size | 8 | | 16 | | 32 | |
|---|---|---|---|---|---|---|---|
| | Kernel size | 3 | 5 | 3 | 5 | 3 | 5 |
| 8 | | 150.9 | 143.7 | **171.2** | 167.5 | 160.8 | 145.0 |
| 16 | | 144.7 | 160.4 | 162.6 | 169.9 | 159.7 | 124.3 |
| 32 | | x | 129.1 | 155.6 | 163.8 | 167.2 | x |

Table 4: Learning rate refinement

| Learning rate | Average reward |
|---|---|
| 1e-3 | 171.2 |
| 1e-4 | 170.4 |
| 1e-5 | **175.5** |

Table 5: Discount rate refinement

| Discount rate | Average reward |
|---|---|
| 0.95 | 175.5 |
| 0.97 | 184.8 |
| 0.99 | **223.8** |

best learning rate (4) and the best discount rate (4). In the following tables I show each combination tried and their corresponding average result values.

### 4.1 Deep Q-Learning

After training the agent over *850* episodes it can be stated that the agent is able to play the game averaging 167 points per game (2, 3). However, as the figures show marginal improvement over the episodes can not be noticed.

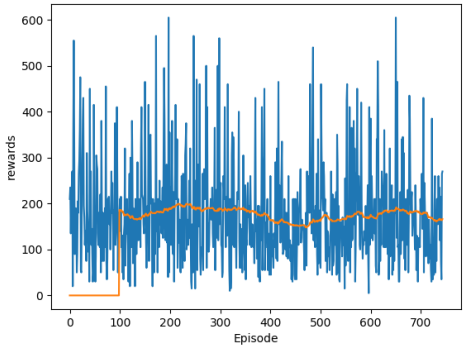


Figure 2: Mean of episode rewards

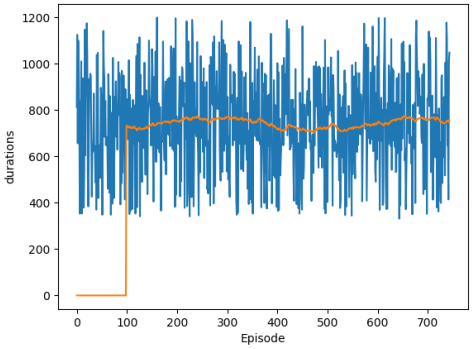


Figure 3: Mean of episode durations

### 4.2 Double Deep Q-Learning

After training the DDQN agent over *850* episodes the results show that it is not performing on a higher level than the simple DQN agent. After the 850 episodes the average reward was a little bit lower than in the other case, 152.4 points on average. We can notice the same tendency throughout the episodes as in the previous episodes, that the average rewards are nearly constant, with a little fluctuation. (4, 5)

## 5 Conclusion

As in the original paper [3] where Double Deep Q-Learning was introduced, it can outperform the classical Deep Q-Learning algorithm by decreasing the overall loss and increasing average episode rewards. In my project I tried to reproduce these results. However, I did a grid search to determine the best hyperparameters for the agents and after finding the optimal results, the DDQN agent was not able to outperform the simple DQN agent. On the contrary, it performed on a slightly worse level.

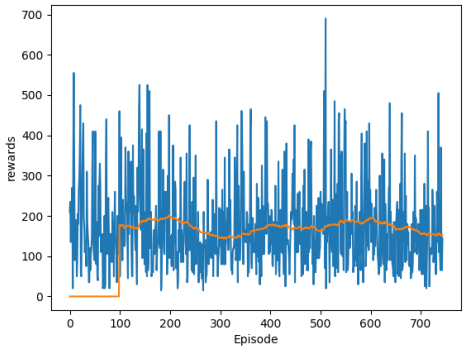


Figure 4: Mean of episode rewards

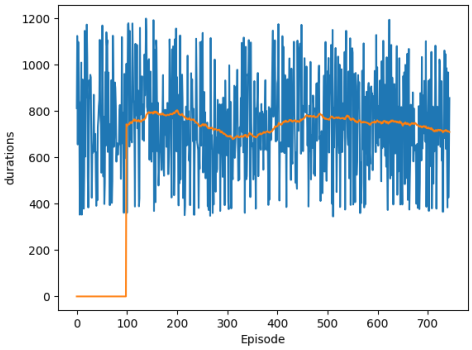


Figure 5: Mean of episode durations

There are several reason which can cause this lower performance. For example, the quality of the preprocessed images are too low, the hyperparameters are not optimal on bigger episode numbers, but only on smaller one or the weight update of the target network is not optimal. These problems can be probably eliminated with further refinement of the hyperparameters.

## References

[1] Bellman, R. (1957). "A Markovian Decision Process". Journal of Mathematics and Mechanics. - `https://www.iumj.indiana.edu/IUMJ/FULLTEXT/1957/6/56038`

[2] Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Martin Riedmiller "Playing Atari with Deep Reinforcement Learning" - `https://arxiv.org/pdf/1312.5602.pdf`

[3] Deep Reinforcement Learning with Double Q-learning Hado van Hasselt and Arthur Guez and David Silver - `https://arxiv.org/pdf/1509.06461.pdf`

[4] OpenAI Gym - `gym.openai.com`