Deep Learning Assignment #2

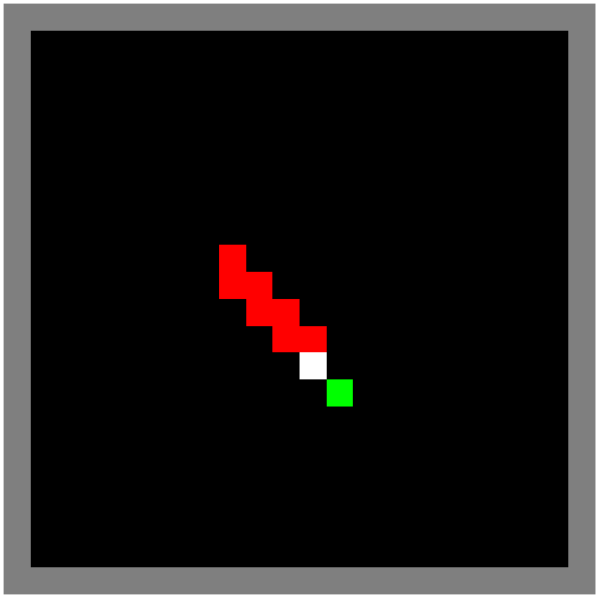
Ruben Belo
MIEI — 55967
NOVA School of Science and Technology
Caparica
rc.belo@campus.fct.unl.pt

João Palma
MIEI — 55414
NOVA School of Science and Technology
Caparica
jp.palma@campus.fct.unl.pt

Andre Matos
MIEI — 55358
NOVA School of Science and Technology
Caparica
afr.matos@campus.fct.unl.pt

Abstract

Deep neural networks are used as function approximators to represent policies or value functions, which is the main concept of Deep Reinforcement Learning. The input to these networks can be raw sensory data, pictures, or other types of data, and the output might be actions or value assessments of the Q-function. For this assignment we are going to train an agent to play the snake game. In this case the input is an image that represents the game board with 20x20 squares as a border of 1 and the output are the 3 values of the Q-function that represents the 3 actions left, forward and right.



Preprint. Under review.

1 Assignment Approach

The working method for this assignment was to train several configurations of the agent in a small number of episodes (around 200).In case of a positive reaction, from any of the configurations, we saved them and trained them up to 500 episodes, selecting the best one to train up to 2000 episodes to get better and more consistent results.

After selecting our fine tuned model we proceeded with an ablation study where we compared our model with and without the target network and replay memory. This final agent also served as the foundation for establishing the double and dueling DQN.

2 Used Heuristics

When we were first trying to get our agent to respond to the environment (the boundary and the fruits), we came up with two distinct heuristics, the first one had the snake move conspicuously in one direction before moving in another to grab the apple. For instance, if the snake was in the lower right corner an the apple in the top right corner the snake would travel to the right until it was underneath the apple and only then would it go up towards the apple.

The other heuristic is more aggressive and made the snake more prone to die by hitting its own body after eating an apple because it made the snake move in a zigzag format towards the apple. After seeing this happening we added a mechanism to allow the snake to move away from its body in some circumstances, but always prioritizing is movement towards the apple.

To generate the instances, we used heuristics no. 1, no. 2, and a combination of the two, where the examples were generated half by one and half by the other. First, we noticed that with the second heuristic, the agent was able to better correlate the apple with its objective, as it couldn't quite eat the apple in the beginning, walking around it or else it would die in the process of going to it. We also noticed that when mixing the two heuristics, the agent didn't learn as well as the other heuristics and didn't get any significant results in the first couple of episodes.

With this in mind, we chose to apply heuristic no. 2 to all of the main models.

3 Exploration strategies

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * np.exp(-decay * episode) (1)$$

$$if \epsilon > \epsilon_{min}, \epsilon* = decay (2)$$

For the tuning of both the ϵ and the decay we tried different two different aproaches. After fine tuning both we arrive at the conclusion that, number 1 decayed faster and in a way was harder to have control of, while number 2 showed to be easier to have control of and for that reason was the one that we chose to use.

4 Deep Q-Network Architecture

For this task, we looked at a variety of architectural theories and put them to the test. The network that provided results more quickly and consistently during all training events is summarized in Figure 1.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 22, 22, 32)	416
activation (Activation)	(None, 22, 22, 32)	0
conv2d_1 (Conv2D)	(None, 22, 22, 32)	4128
activation_1 (Activation)	(None, 22, 22, 32)	0
conv2d_2 (Conv2D)	(None, 22, 22, 64)	8256
activation_2 (Activation)	(None, 22, 22, 64)	0
max_pooling2d (MaxPooling2D)	(None, 11, 11, 64)	0
features (Flatten)	(None, 7744)	0
dense (Dense)	(None, 256)	1982720
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 3)	195

Total params: 2,036,867
Trainable params: 2,036,867
Non-trainable params: 0

Figure 1: Architecture used in this assignment

Before reaching this network, we modify the number of convolutional layers (1, 2, or 3), as well as the kernel size and the number of filters for each layer. For the dense layers, we test two strategies: one in which we change the number of neurons while maintaining their constant number throughout all layers, and the other in which the number of neurons decreases from layer to layer.

4.0.1 Target Network

When we trained our agent with the Target Network, we discovered that we should train the model every four training steps and update the Target Network when an episode ends, however only 100 steps out of 100 produced the greatest results.

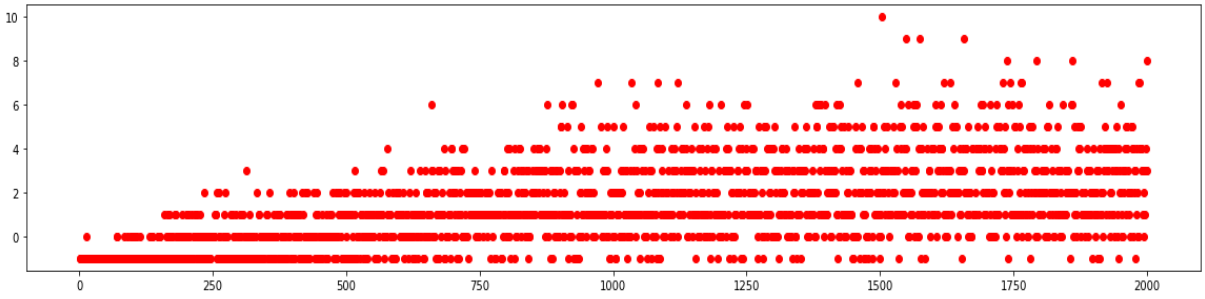


Figure 2: Score while training with 2000 episodes

4.1 Double DQN

4.1.1 Implementation

The Double DQN implementation employs DQN architecture with a Target Network, but with the appropriate alterations to reduce overestimation, so that the online network selects the action with the greatest Q-values while the Target Network estimates the Q-value of this action.

The target Q-values for a particular state-action combination are updated in DQN with Target Network as follows:

$$QTarget(s, a) = Reward + DiscountFactor * max(QTarget(s', a'))$$

While the Double DQN is being updated as follows:

$$QTarget(s, a) = Reward + DiscountFactor * QTarget(s', argmax(QOnline(s', a')))$$

4.1.2 Standart

Here we can see that with the Double DQN we get slightly faster to a close value for the Q function compared to our default DQN. However given 2000 episodes the final results for both the DQN and the DDQN are very similar to one another.

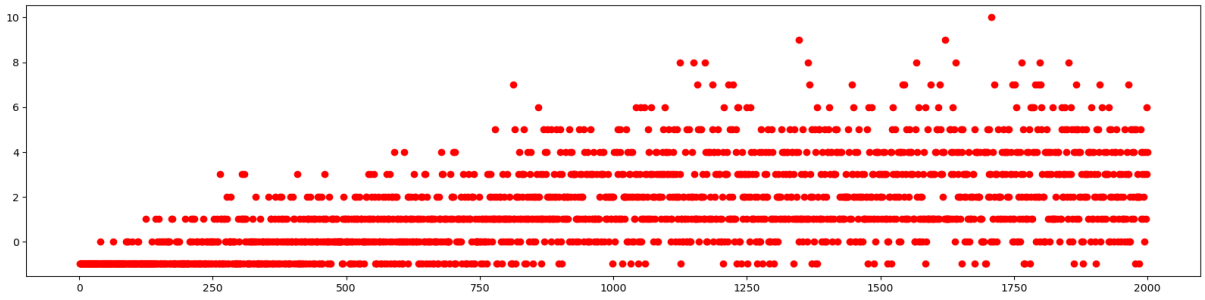


Figure 3: Score while training with 2000 episodes the Double DQN

4.1.3 Standart with Heuristic no. 1

Like we mentioned previously the main characteristic of the Double DQN also reflects when we apply it to the heuristic 1, even showing success in reaching the score value of 12. This means that even thought we initially arrived to a conclusion that for a small amount of episodes the best heuristic is H2 this H1 also presents very similar results when we have more episodes. Perhaps there is still more studying and testing to be done regarding what is the best heuristic for our scenario and how important if important at all the heuristic is.

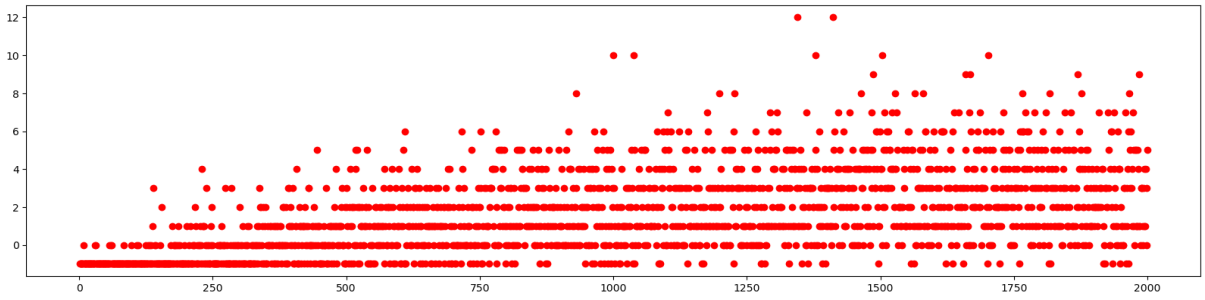


Figure 4: Score while training with 2000 episodes the Double DQN with Heuristic 1

4.1.4 Dueling plus Double DQN

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 22, 22, 3)]	0	[]
conv2d (Conv2D)	(None, 22, 22, 32)	416	['input_1[0][0]']
conv2d_1 (Conv2D)	(None, 22, 22, 32)	4128	['conv2d[0][0]']
conv2d_2 (Conv2D)	(None, 22, 22, 64)	8256	['conv2d_1[0][0]']
max_pooling2d (MaxPooling2D)	(None, 11, 11, 64)	0	['conv2d_2[0][0]']
features (Flatten)	(None, 7744)	0	['max_pooling2d[0][0]']
dense_4 (Dense)	(None, 256)	1982720	['features[0][0]']
dense (Dense)	(None, 256)	1982720	['features[0][0]']
dense_5 (Dense)	(None, 128)	32896	['dense_4[0][0]']
dense_1 (Dense)	(None, 128)	32896	['dense[0][0]']
dense_6 (Dense)	(None, 64)	8256	['dense_5[0][0]']
dense_2 (Dense)	(None, 64)	8256	['dense_1[0][0]']
dense_7 (Dense)	(None, 3)	195	['dense_6[0][0]']
dense_3 (Dense)	(None, 1)	65	['dense_2[0][0]']
add (Add)	(None, 3)	0	['dense_7[0][0]', 'dense_3[0][0]']

Total params: 4,060,804
Trainable params: 4,060,804
Non-trainable params: 0

Figure 5: Dueling Architecture used for this test

Using the before mentioned design, we chose to stack the Dueling DQN implementation on top of the Double DQN implementation in order to assess the outcome of combining the 2 strategies. Using both of these approaches together yields poor results, at least when training for 2000 episodes.

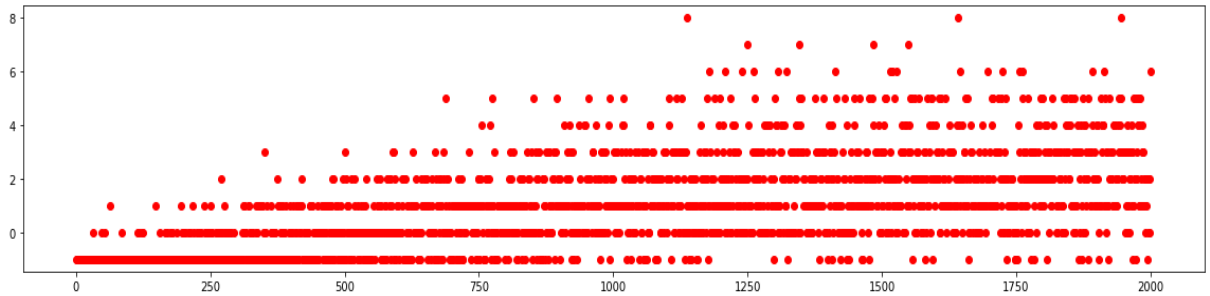


Figure 6: Score while training with 2000 episodes

4.1.5 Double DQN plus enviermentwith 3 apples

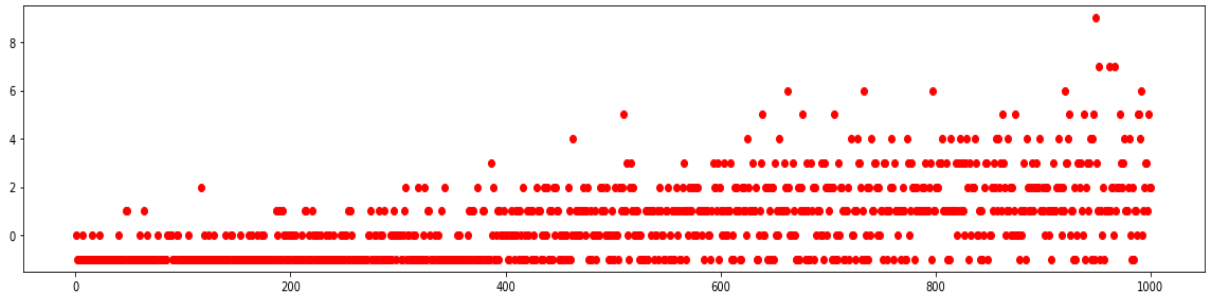


Figure 7: Score while training with 1000 episodes

5 Ablation Study

When the replay memory was removed, the model only trained with the game it was playing, which means it trained with each action it took. This was insufficient data for the model to learn because these types of models, particularly those with imagens as input, require a large volume of data to provide good results.

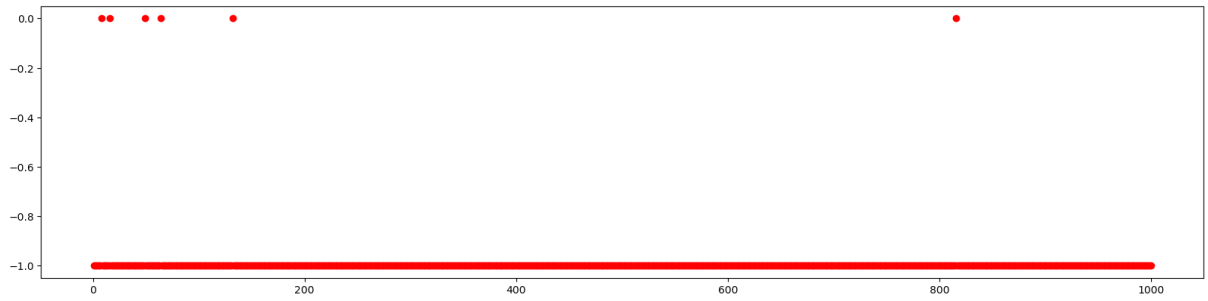


Figure 8: Score while training with 1000 episodes without replay memory

On the other hand, when we removed the Target Network, it learned and obtained nearly the same scores but was less consistent; this was the expected result because the Target Network is used to estimate the Q-value and because it does not change in each iteration of the training, it helps to stabilize the learning process.

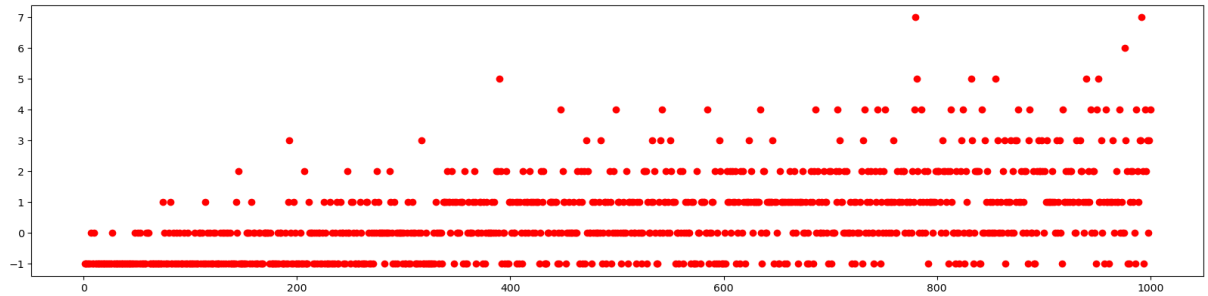


Figure 9: Score while training with 1000 episodes without target network

Finally when neither the Target Network nor the Replay Memory was used the result was clearly negative showing almost no progess at all during the 1000 episodes, showing how important it is to use these techniques.

6 Complications and problems that we faced

One of our biggest problems that we faced was the amount of time needed to see any change that we made, since the agent takes a lot of time to train and to see if it had any improvements. This happens specially if you don't have a gpu, which was the case with one of us, and it was simply impossible to train the agent with more than 500 episodes, limiting the amount of tests we could make and things we wanted to try to improve. We also tried to use Google Colab to handle this issue, and it was going faster than our computers with gpu, but after 2 hours we would get out of available runtime and it would go down. Reinforcement Learning is really data hungry and takes a lot of time for people who don't have really good computers, not making accessible for everyone. So whenever the proposed solution is related with Reinforcement Learning we should always ask ourselves, even before starting, how fast do we need the solution and in what environment will we be working.

7 Next Steps

The network and implementation of Double DQN, Dueling DQN, DQN, and Double + Dueling + Double DQN would need to be tested with episode numbers in the tens of thousands or more in order to further improve the results and the analysis of the effect of settings, heuristics, and whether their use is even necessary. When it happens, the findings may be conclusive.

8 Bibliography

https://youtu.be/-NJ9frfAWRo

https://medium.com/@hugo.sjoberg88/using-reinforcement-learning-and-q-learning-to-play-snake-28423dd49e9b

https://www.geeksforgeeks.org/ai-driven-snake-game-using-deep-q-learning/

https://towardsdatascience.com/snake-played-by-a-deep-reinforcement-learning-agent-53f2c4331d36

https://ap.ssdi.di.fct.unl.pt/teoricas.html

https://claudia-soares.notion.site/Deep-Learning-Home-4613aed8bdce4b7db6255330966a7480

https://medium.com/analytics-vidhya/implementation-of-dqn-double-dqn-and-dueling-dqn-with-keras-rl-2020-bc55339b21a9