Aprendizagem Profunda Aplicada – 2022/2023

Assigment 3

Duarte Sousa Cruz
2017264087

Gonçalo Tomaz Arsénio
2017246034

December 22, 2022

1 Introduction

Our goal with this assignment is to understand how Deep Reinforcement Learning (DeepRL) works and how to apply these techniques. In this work we will explore and analyse Deep Q-Learning, Double Deep Q-Learning and Dueling Deep Q-Learning algorithms.

In the CarRacingv2 environment, a car must race through a track and can be controlled by discrete or continuous actions. To control the car in discrete mode, 5 actions are available ('nothing', 'left', 'right', 'gas', 'brake'). In continuous mode, 3 different variables can be changed: Steering wheel, Throttle and Brake. When using the previously mentioned Deep Q-learning methods, the action values must be discretized into a set of appropriate actions. The target track and the colours of the environment are randomly generated at each episode. The reward is -0.1 for each image and +1000/N for each newly visited track tile, where N is the total number of tiles visited on the track. In practice, this means that the car will receive a positive reward each time it finds a new tile when it enters the tile, otherwise a value of -0.1 will be assigned. An additional reward of -100 is awarded if the car moves far away from the track. The default conditions for completing the game are an episode length of more than 1000, all tiles on the track are visited, or the car moves too far away from the track.

2 Part I

For this first part, we need to implement a Convolutional Neural Network (CNN) that receives as input an RGB image (or a stack of RGB images) corresponding to the visual representation of the OpenAI CarRacingv2 environment. The target condition is to obtain a total reward per episode higher than 200 in the previous 20 episodes (in one of the control modes, discrete or continuous).

2.1 Deep Q-Learning

Deep Q-Learning uses the idea of Q-Learning to approximate the optimal Q function and in this way maximize the reward achieved by the agent through an optimal policy, and goes one step further. Instead of a Q-table, we use a neural network that takes a state and approximates the Q-values for each action based on that state. So now we use a deep neural network that takes the state as input and generates different Q values for each action. Then we again choose the action with the highest Q-value. The learning process is still the same with the iterative update approach, but instead of updating the Q-Table, here we update the weights in the neural network so that the outputs are improved.[2]

The Q function in DQN is given by:

$$Q(s,a;\theta) = r + \gamma \underset{a'}{max}\, Q(s',a';\theta') \tag{1}$$

where θ represents the policy network weights and θ' the target network weights.

2.1.1 CNN Architecture

For the implementation of the *DQN* we used a convolutional neural network based on [1].

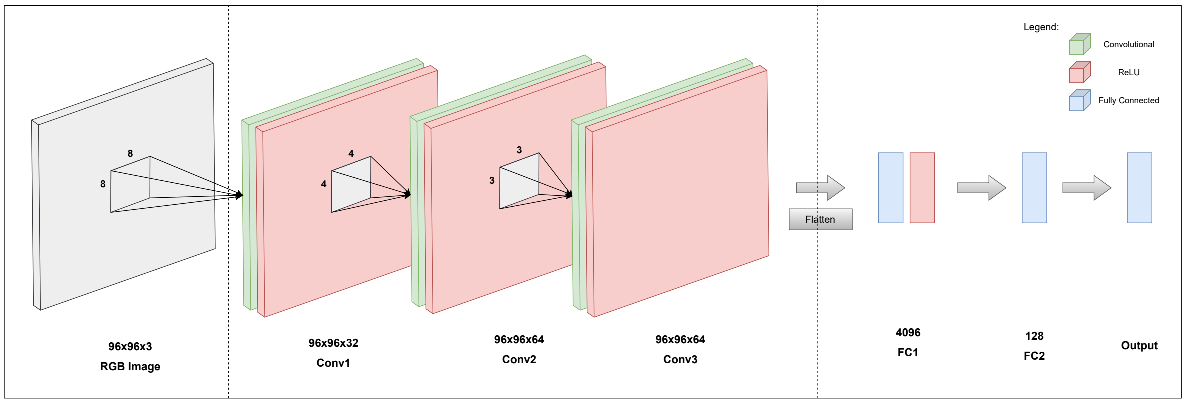

Figure 1: Architecture of the convolutional neural network for DQN

The outputs correspond to the predicted Q-values of each action for the input state. The main advantage of this type of architecture is the ability to compute Q-values for all possible actions in a given state with only a single forward pass through the network. The input to the neural network consists of a 96x96x4 image generated by the OpenAI CarRacingv2 simulation environment. The first hidden layer convolves 32 filters of 8x8 with Stride 4 with the input image and applies a rectified linear unit (ReLU) activation function. The second hidden layer convolves 64 filters of 4x4 with Stride 2, again followed by the ReLU. This is followed by a third convolutional layer that convolves 64 filters of 3x3 with Stride 1, followed by the rectifier. After that, we have two fully-connected layers consisting of 4096 and 128 rectifier units, respectively. The output layer is a fully connected linear layer with a single output for each valid action. The number of valid actions depends on the mode we use (continuous or discrete).

2.2 Double Deep Q-Learning

The architecture used for the Double DQN is the same implemented for DQN. The difference is the Q function given by:

$$Q(s,a;\theta) = r + \gamma\, Q(s', \underset{a'}{argmax} Q(s',a';\theta);\theta') \tag{2}$$

In Double DQN, at each step, the value of all action-value combinations for all possible actions in the given state is read from the policy Q-network, which is continuously updated. Then an argmax is drawn over all state-action values of these possible actions and the state-action combination that maximizes the value is selected. To update the policy Q-network, the corresponding target value of such selected state-action combination is taken from the target Q-network, as we can see in the Fig. 2.Double DQN algorithm proposes to solve the problem of 'overestimation' of the Q-values while avoiding the instability of the target values.

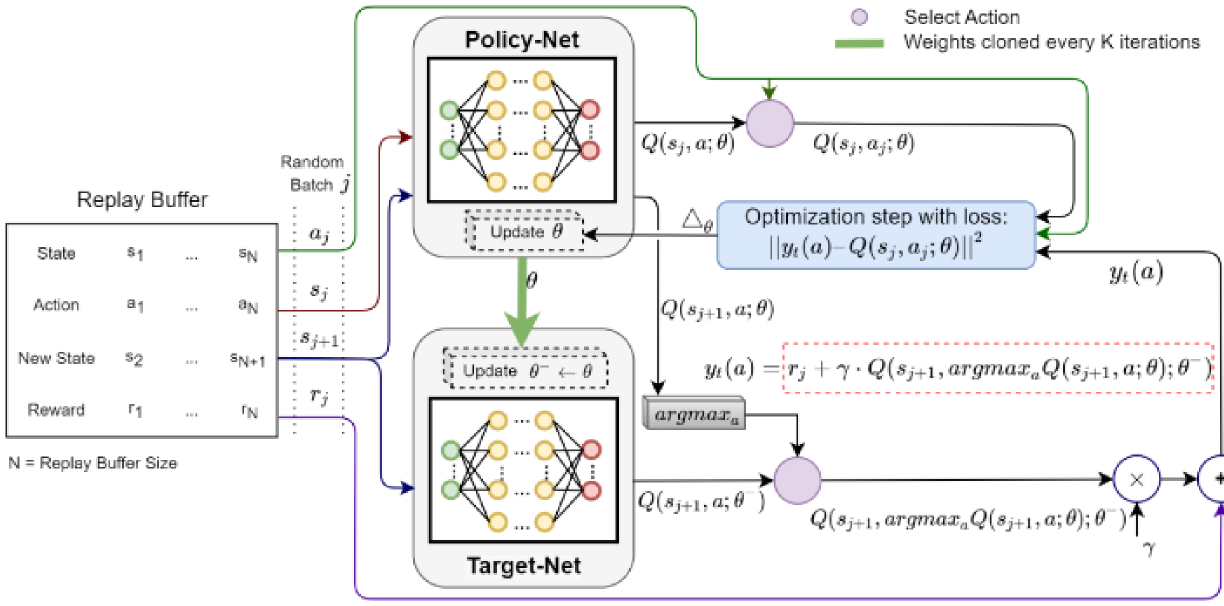
Select Action
Weights cloned every K iterations
Policy-Net
Target-Net
Replay Buffer
State s_1 ... s_N
Action a_1 ... a_N
New State s_2 ... s_{N+1}
Reward r_1 ... r_N
N = Replay Buffer Size
Random Batch j
a_j
s_j
s_{j+1}
r_j
Update θ
θ
Update $\theta^- \leftarrow \theta$
$Q(s_j, a; \theta)$
$Q(s_j, a_j; \theta)$
$\triangle_\theta$
Optimization step with loss: $||y_t(a) - Q(s_j, a_j; \theta)||^2$
$y_t(a)$
$Q(s_{j+1}, a; \theta)$
$y_t(a) = r_j + \gamma \cdot Q(s_{j+1}, argmax_a Q(s_{j+1}, a; \theta); \theta^-)$
$argmax_a$
$Q(s_{j+1}, a; \theta^-)$
$Q(s_{j+1}, argmax_a Q(s_{j+1}, a; \theta); \theta^-)$
γ

Figure 2: Double Deep Q Network[7]

2.3 Dueling Deep Q-Learning

The Dueling DQN algorithm proposes that the same neural network splits its final layer into two parts, one estimating the state value function for state s (V(s)) and the other estimating the advantage function for each action a (A(s, a)), and at the end combine both parts into a single output that estimates Q-values. This change is helpful because sometimes it is not necessary to know the exact value of each action, so in some cases it is sufficient to learn only the state value function. The Q function is given as:

$$Q(s, a) = V(s) + A(s, a) \tag{3}$$

2.3.1 CNN Architecture

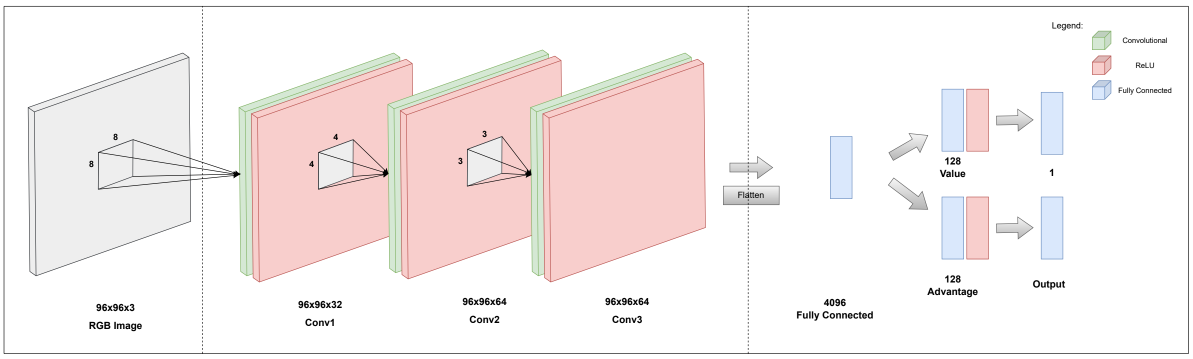

Figure 3: Architecture of the convolutional neural network for Dueling DQN

In Dueling DQN, we have a non-sequential Deep Learning architecture where the model layers branch into two distinct streams (sub-networks) after the convolutional layers, each with its own fully connected layer and output layers. The first of these two branches corresponds to that of the value function and has a single node in its output layer. The second branch is for the advantage network and calculates the value of the advantage of a particular action over the base value of the current state. From fig. 3 we can see the division of the last layer. By adding this kind of structure allows the network, to better distinguish actions and significantly improve learning. We can also verify that the first part of the CNN is the same as in the DQN.

2.4 Hyper Parameters

We changed some parameters to improve the performance of our model. Due to the limited use of Colab and the computational time required to test each model, we did not perform many parameter tests. Ultimately, we increased the number of control steps, i.e., the number of iterations in which the same action is performed in the environment, and the frame stack, i.e., the number of consecutive frames used to represent the state, to 3, and since we were able to have a total reward higher than 200 the previous 20 episodes, every thing else remain the same.

2.5 Results

In table 1 and 2 we have the total rewards for 10 different scenarios, 5 with random tracks and 5 with random tracks and random scenarios. These scenarios were run with the bestNet from each Deep RL algorithm.

Table 1: **Continuous** validation score: 1-5 Random Tracks, 6-10 Random Tracks and Environments

Scenario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
DQN	396.37	319.86	231.99	164.13	338.20	550.17	10.51	-84.22	297.64	-38.55
Double DQN	593.95	778.47	841.17	307.96	522.02	41.42	-71.93	-76.29	-13.69	-93.35
Dueling DQN	864.78	882.77	840.32	937.89	882.85	-43.46	-46.76	-35.11	26.89	-60.82

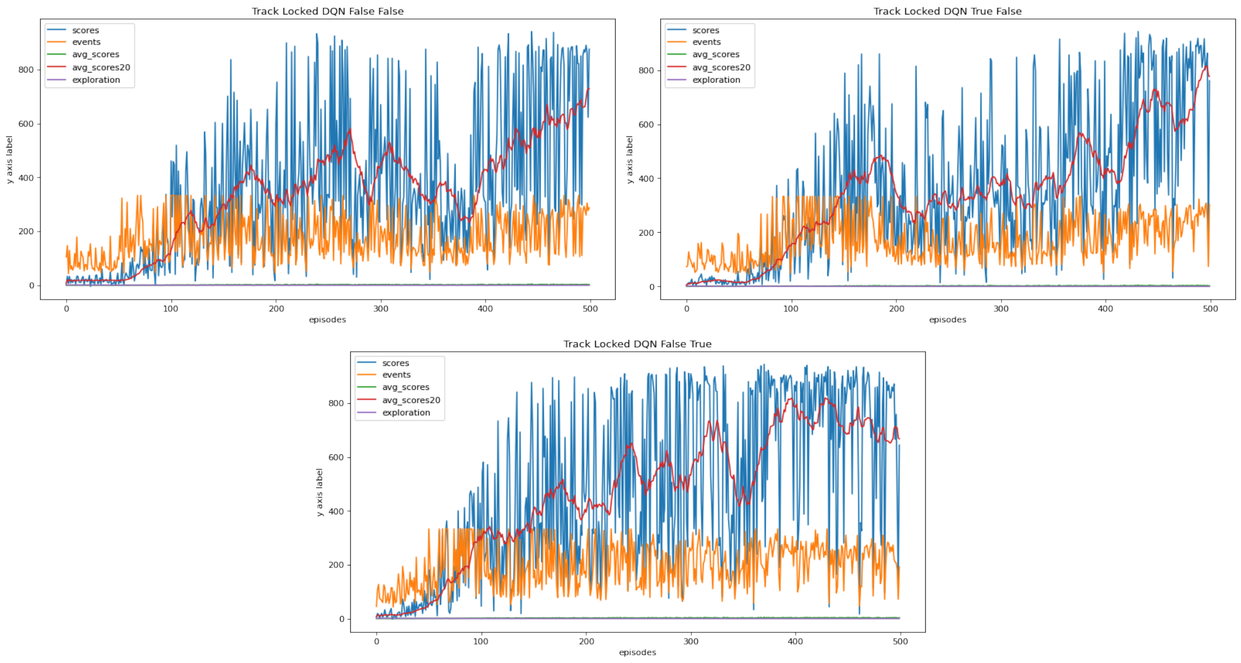

Figure 4: Results of the training for **continous** mode: DQN (left), Double DQN (right) and Dueling DQN (center)

From the table 1 and fig. 4 we can see that the double DQN has a slightly better performance than the DQN. This is because, as mentioned earlier, the DQN uses the same network to both select the best action and evaluate the expected rewards for each action. This can lead to an overestimation of expected rewards, which in turn can lead to suboptimal performance. DDQN addresses this problem by using two separate networks: one network is used to select the best action, while the other network is used to evaluate the expected rewards for each action. This decoupling of action selection and reward estimation helps reduce the overestimations that can occur in DQN models, leading to better performance.

However, Dueling is the model that gives us the best performance. This is due to the use of a special architecture, explained in Section 2.3, which allows the model to better focus on the important aspects of the problem, resulting in better performance.

We can also see that our bestNet for any deep reinforcement algorithm performs poorly in scenarios with random environments because they trained in the same environment during all the episodes and then have to do a track in a completely different environment with random colours.

Table 2: **Discrete** validation score: 1-5 Random Tracks, 6-10 Random Tracks and Environments

Scenario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
DQN	539.04	846.76	866.35	753.45	863.56	-70.67	186.31	362.09	-92.59	297.88
Double DQN	568.91	865.44	693.64	918.69	596.89	-39.70	-75.37	45.07	184.52	672.37
Dueling DQN	630.06	183.13	460.51	870.95	449.96	-85.98	-79.38	384.07	-49.39	-72.99

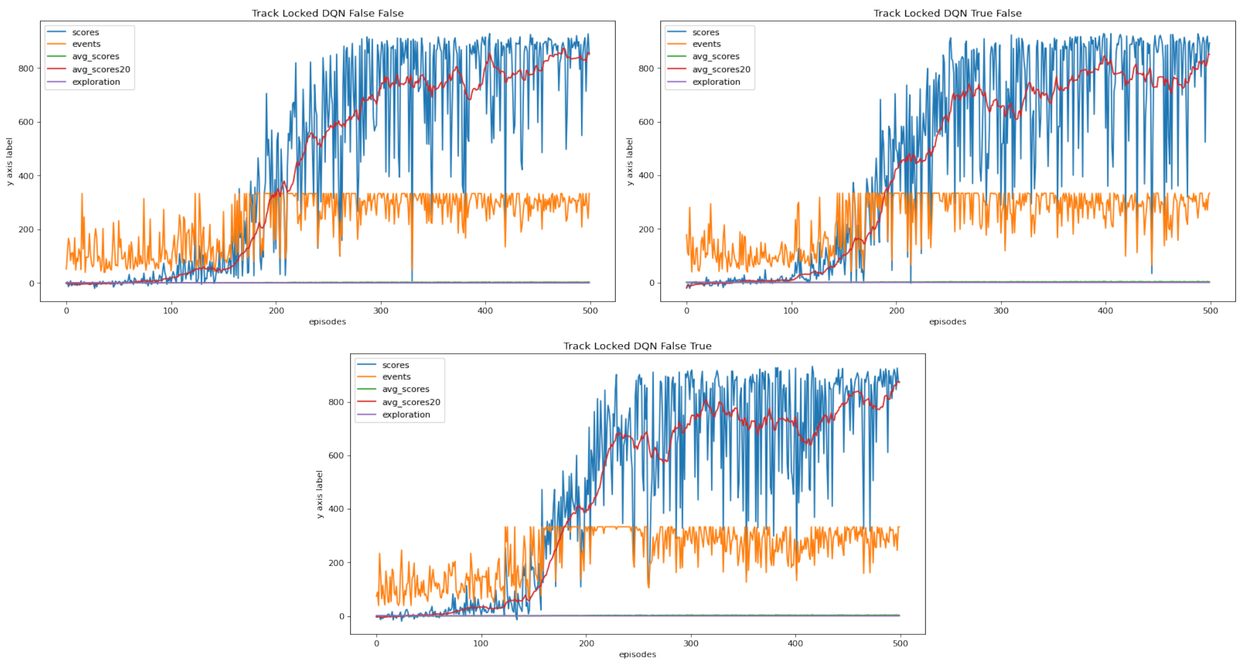

Figure 5: Results of the training for **discrete** mode: DQN (left), Double DQN (right) and Dueling DQN (center)

If we analyze the results from the table 2 and the figure 6, we can observe the performance increase of all algorithms, and they have more similar performance than when using continuous mode, but dueling is still our best model. Using discrete actions enlarged our scores because using a discrete action space means that the possible actions the agent can take are limited to a fixed set of possibilities. For example, the agent can accelerate, brake, or turn left or right. In contrast, a continuous action space would allow the agent to perform any action within a certain range, such as applying a specific amount of acceleration or turning the steering wheel a certain amount.

In general, using a discrete action space can make it easier for a deep Q-learning algorithm to converge on the optimal policy, by reducing the number of possible actions the agent can take at each step. This can help the algorithm learn more efficiently, since it does not have to sift through as many possible actions to find the best one. Also, using a discrete action space can make the definition of the reward function easier, since the possible actions are known in advance and can be explicitly defined in the function.

3 Part II

In the second part of the task, we will use the best performing method from part one, in our case the Discrete Dueling Deep Q-Network, and first train it from scratch with random environments, i.e., the colours of the track and the grass are randomly generated, and then compare it to training with Transfer Learning by loading the weights of the best method from part one into our network trained in random environments.

3.1 Results

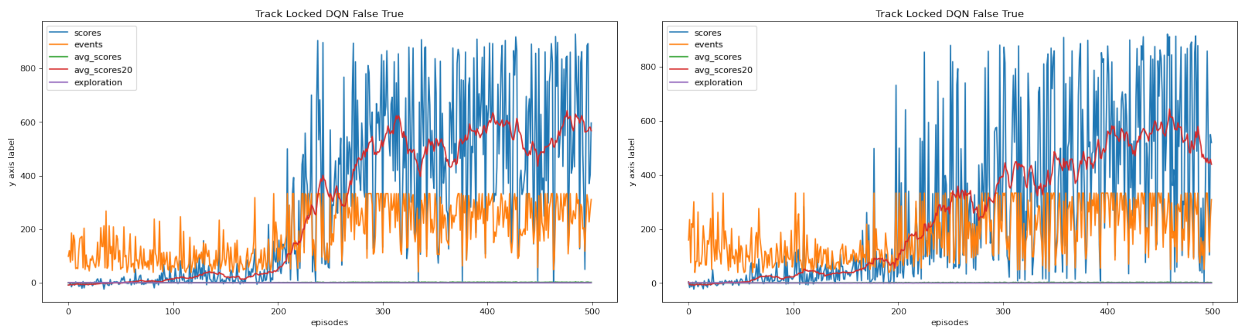

Figure 6: Results of the training in Random Environment: From Scratch (left) and with Refining (Right)

When the model is trained on the car racing environment, it learns to map observations of the environment (i.e., the state of the car and its surroundings) to actions that maximize cumulative reward. However, since the environment is different for each episode, the DQN may not be able to generalize its knowledge to all possible environments, which could limit its performance. Therefore, we see a decrease in the average score compared to the model in the first part, which was trained with a "fixed" environment.

One way to address this problem is to use a large number of training episodes so that the DDQN is exposed to a wide range of environments during training. This allows the DDQN to learn to generalize its knowledge to different environments, which can improve its performance.

When using transfer learning to fine-tune a previously trained model, it is exposed to a different environment than the one in which it was originally trained. This allows the DQN to learn to generalize its knowledge to different environments, which can improve its performance.

However, the effectiveness of transfer learning in this situation also depends on the quality of the pre-trained model used for refinement. Since the pre-trained model was trained in a fixed environment and the model to be refined is trained in a random environment in each episode, the pre-trained model may not contain relevant information for the new environment. In this case, fine-tuning the previously trained model in the new environment may not result in better performance.

4 Part III

4.1 Describe the current reward model of the CarRacing-v2 environment and highlight some of the problems this reward may impose.

The current reward model is based on give points or take consonant the behavior of the model. At every frame, the model gives -0.1 points and +1000/N for every new track tile visited, where N is the total number of tiles visited in the track. When the car goes far from the track it receives a penalty of -100, that causes the agent to limit its movements due to the large reward penalty, preventing exploration and attempting the track.

The default termination conditions are an episode length greater than 1000, all tiles in the track are visited, or the car gets too far away from the track.

This reward model has a few potential problems. First, it rewards the agent only for visiting track tiles, but not for other important factors such as finishing the track quickly, avoiding getting out of the track or spinning. This means that the agent may focus only on visiting as many tiles as possible, without considering other aspects of the task that may be important for successful performance.

Second, the reward is heavily discounted over time, with the agent receiving -0.1 points for every frame. This means that the agent will be incentivized to finish the track as quickly as possible, in order to maximize its reward. However, this may lead the agent to take risky actions that may result in losing control or other failures, which could ultimately reduce its overall reward.

Overall, this reward model may not provide the agent with the right incentives to learn to successfully complete the track. A better reward function would take into account a wider range of factors, such as the speed and safety of the agent's actions, as well as its ability to successfully navigate the track and reach the finish line.

4.2 Comparing the performances with DQN, Double DQN and Dueling DQN. How can you explain the gaps in performance (if any)?

Concluding everything that has already been said throughout the report, the performance of DQN, Double DQN, and Dueling DQN can vary depending on the specific task and environment in which they are used. DQN is typically the most basic and can struggle with long-term dependencies, while Double and Dueling can improve on this limitation to some extent. However, both can be more complex to implement and may require more computational resources, so they may not always be the best choice for a given problem.

4.3 Without additional changes, training for a larger number of episodes would likely yield further improvements in performance?

Yes, that is correct. In general, the more training episodes a deep reinforcement learning network undergoes, the more it can improve its performance. However, there are limits to how much performance can be gained through additional training, and at some point the network will reach a plateau where further training will not yield significant improvements. Additionally, the amount of time and computational resources required to train a network for a larger number of episodes can be significant, so it's important to consider the trade-offs.

4.4 Do you consider that the best model (as defined in the provided Google Colab notebook) is the true best model?

It might not be, it depends on how that model was trained. The model may have had a better score because it was trained on easier tracks, however this does not reflect the general picture and on other tracks it may no longer be the best model. With this, we consider that the best model does not always turn out to be the true best model.

4.5 Describe the importance of exploration and exploitation in the learning process

Exploration and exploitation are both important in the learning process of a deep reinforcement learning network. Exploration refers to the process of trying out different actions in a given environment to gather information and learn more about the consequences of those actions. This allows the network to improve its understanding of the environment and make better decisions in the future. Exploitation, on the other hand, refers to the process of using the information that has been gathered through exploration to make the most optimal decision in a given situation. This allows the network to maximize its rewards and achieve its goals more efficiently.

The balance between exploration and exploitation is often a key factor in the learning process. If an agent focuses too much on exploration, it may not make use of its existing knowledge and may not perform as well in the short term. On the other hand, if an agent focuses too much on exploitation, it may not discover new strategies or adapt to changes in the environment, leading to suboptimal performance in the long term. Finding the right balance between exploration and exploitation is therefore an important aspect of the learning process.

4.6 Explain the influence of the replay buffer, could we further improve performance by modifying the replay buffer?

The replay buffer is a collection of experience tuples (state, action, reward, next_state) that are gradually stored to the buffer as we interact with the Environment. The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay.

The basic idea behind experience replay is storing past experiences and then using a random subset of these experiences to update the Q-network, rather than using just the single most recent experience.

There are a number of ways in which the replay buffer can be modified in order to further improve performance. One common technique is to use a prioritized replay buffer, in which experiences are prioritized based on how important they are for learning. This can help the agent focus on the most informative experiences and can lead to faster learning. Other techniques include using different sampling methods to select experiences from the replay buffer, or using techniques like "herding" to select a more representative set of experiences.

Overall, the replay buffer is an important component of many deep reinforcement learning algorithms, and modifying it can be a powerful way to improve the performance of the agent. However, it is important to carefully consider the trade-offs of different replay buffer modifications and choose the one that is most suitable for a specific task and environment.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518:529–33, 02 2015.

[2] Loeber, P. (2022, February 10). Reinforcement learning with (deep) Q-learning explained. News, Tutorials, AI Research. Retrieved December 15, 2022, from https://www.assemblyai.com/blog/reinforcement-learning-with-deep-q-learning-explained/

[3] Ausin, M. S. (2020, November 25). Introduction to reinforcement learning. part 4. double DQN and dueling DQN. Medium. Retrieved December 15, 2022, from https://markelsanz14.medium.com/introduction-to-reinforcement-learning-part-4-double-dqn-and-dueling-dqn-b349c9a61ea1

[4] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015)

[5] Moghadam, P. H. (2019, July 23). Deep reinforcement learning: DQN, double DQN, dueling DQN, noisy DQN and DQN. Medium. Retrieved December 15, 2022, from https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823

[6] M. Sewak, "Deep Q Network (DQN), Double DQN, and Dueling DQN",Deep Reinforcement Learning, pp. 95-108, Springer, 2019.

[7] Image from pratical lessons given by Prof. Luís Garrote

[8] Van Hasselt, H., Guez, A., &amp; Silver, D. (2016). Deep reinforcement learning with double Q-learning. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). https://doi.org/10.1609/aaai.v30i1.10295

[9] Yang, A. (2022, July 25). What is exploration vs. exploitation in reinforcement learning? Medium. Retrieved December 16, 2022, from https://medium.com/@angelina.yang/what-is-exploration-vs-exploitation-in-reinforcement-learning-a3b96dcc9503