1 Introduction

In this assignment it is expected that you learn and understand how to apply Deep Reinforcement Learning (DeepRL) techniques to control an agent in a simulation environment. This assignment will focus on image-based state representation. This assignment is part of your continuous evaluation. You should provide a report and notebooks (.ipynb), which include your development and analysis for the requested tasks.

2 Simulation Environments

In this assignment, the **CarRacing-v2** simulation environment from the OpenAI gym and a custom simulation environment will be explored. In the **CarRacing-v2** environment, a car must race through a track and can be controlled by discrete or continuous actions. The state representation is composed of a 96×96 RGB image. To control the car in discrete mode 5 actions are available ('nothing','left','right','gas','brake'). In continuous mode, 3 different variables can be modified: steering wheel, gas, and brake (e.g., [-1,0.4,0.1]). The steering wheel controls the car's steering direction (range: [-1,1]), the gas simulates the accelerator in the car (range: [0,1]), and brake corresponds to the brake pedal (range: [0,1]). While using Deep Q-Learning based methods [1, 2, 3], the action values must be discretized into a set of suitable actions. The goal track and environment colors are randomly generated at every episode (see **UseRandomize**). The reward is -0.1 at every frame and +1000/N for every new track tile visited, where N is the total number of tiles visited in the track. In practical terms, every time the car finds a new tile it generates a positive reward when entering the tile, otherwise a -0.1 reward value is given. An additional reward of -100 is given when the car goes far from 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. For the custom simulation environment, the goal is to drive a mobile robot (differential kinematics) while avoiding obstacles. By default, 3 actions are available, go forward, turn left and go forward and turn right and go forward. The actions can be modified, if necessary, to improve the agent's behaviors. The observations can be configured using the flags available in **Modes**. For this assignment only Modes.PoseOnly and Modes.PoseAndLocalMapInline are relevant. Modes.PoseOnly returns a vector containing three values, two values corresponding to the normalized direction to the goal and one for the agent's orientation. Modes.PoseAndLocalMapInline returns a vector that includes the aforementioned values and includes a flattened version of a local map. The local map is centered on the agent, and its dimensions can be configured when the environment is initialized. The reward is the negative distance to the goal (multiplied by a scaling factor). When the agent collides it return a high negative reward and receives a high positive reward when the agent reaches the goal.

3 Theoretical background

3.1 Q-Learning

The Q-Learning algorithm [4], is an iterative Reinforcement Learning (RL) method used to calculate the Q function (or Q-Value - $Q(s, a)$) of a given state-action pair (a, s) which represents the expected return of taking the action a when in state s. The Q-Learning goal is to maximize the value of the Q function by incorporating multiple events gathered through exploration and exploitation in the environment. Considering the optimal Q function ($Q^*(s, a)$), the optimal policy ($\pi^*(s)$) can be extracted by selecting the action that maximizes the reward,

$$\pi^*(s) = arg\max_{a'} Q^*(s, a') \tag{1}$$

The goal in Q-Learning is to approximate the optimal Q function and in that way, maximize the reward obtained by the agent by using an optimal policy. To obtain an approximation of the optimal Q function ($Q^*(s, a)$), the Q-Learning equation is used to recursively update the Q function $Q(s, a)$, and is given by:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{Bellman equation}}] \tag{2}$$

It uses the Bellman equation which represents the maximum future reward given by the current reward signal r and the maximum future reward $\max_{a'} Q(s', a')$, γ is the discount factor and α is the learning rate. The discount factor influences the importance of the future rewards, while the learning rate conditions the Q function assimilation of new events. A lookup table (Q-Table) is commonly used to store the maximum expected future rewards for each possible state-action pair.

3.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DeepRL) [5] can be seen as an extension of RL where a deep learning architecture is combined with RL algorithms. By using a deep learning architecture to approximate a nonlinear function between inputs and outputs, policies can be approximated. With the help of deep learning architectures, high-dimensional states that were previously intractable with RL algorithms can now be solved with DeepRL. One important characteristic of using DeepRL over RL is the generalization capabilities found with DeepRL [6]. DeepRL models can correctly output actions for unseen inputs by recalling similar states that were seen during training. This characteristic allows DeepRL models to adapt to different scenarios from the ones used in training if similar features can be found in the new scenarios. Three methods inspired in Q-Learning will be addressed: Deep Q-Learning (DQN) [1], Double Deep Q-Learning (Double DQN) [2], Dueling Deep Q-Learning (Dueling DQN) [3].

3.2.1 Deep Q-Learning

The DQN [1] algorithm combines the Q-Learning algorithm with deep neural networks. Since deep neural networks are suitable as nonlinear function approximates, the goal with DQN is to approximate the Q function ($Q(s, a)$) replacing the previously used Q-Tables that were not suitable for high-dimensional problems. Two deep neural networks are used to approximate the Q function, the policy network, and the target network. The policy network estimates the Q-Values for the current state-action pair ($Q(s, a)$) while the target network (sharing the same architecture as the policy network) estimates the Q-Values for the next state-action pair ($Q(s', a')$). Learning occurs only on the policy network and the weights of the policy network are cloned to the target network at defined update intervals. This architecture with two deep neural networks allows for a stable learning process due to the weight freezing in the target network.

The Q function in DQN is given by:

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta') \tag{3}$$

where θ represents the policy network weights and θ' the target network weights. To train a deep neural network architecture, the loss function is defined as:

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta)\right)^2\right] \tag{4}$$

where $\mathbb{E}[\cdot]$ denotes the expected value of a random variable given the followed policy. In the DQN algorithm presented in [1] an experience replay buffer is used in order to avoid strong input correlations due to consecutive samples. For each episode, the tuple (s, a, r, s') is added to the buffer. At each batch update of the policy network, N_b tuples are uniformly sampled from the experience replay buffer and used to update the policy network.

3.2.2 Double Deep Q-Learning

Double Deep Q-Learning [2] is inspired by the Double Q-Learning algorithm [7] and improves DQN by reducing overconfident estimations. Compared with the approach in DQN, in the Double Deep Q-Learning the Q function in the target network is given by:

$$Q(s, a; \theta) = r + \gamma Q(s', arg\max_{a'} Q(s', a'; \theta); \theta') \tag{5}$$

.

3.2.3 Dueling Deep Q-Learning

Dueling Deep Q-Learning [3] proposes a different approach to compute the Q function ($Q(s,a)$). In Dueling Deep Q-Learning, the Q function is given as,

$$Q(s,a) = V(s) + A(s,a) \tag{6}$$

where $V(s)$ is the value function and represents the reward to be collected at state s, and $A(s,a)$ is the advantage function representing the valuation of the state-action pair. In practice, compared to the DQN, the policy network splits at the end into two branches: value, and advantage. However, the aforementioned formulation is non-identifiable as given the value of $Q(s,a)$, there are an infinite number of combinations of $V(s)$ and $A(s,a)$ that lead to that value. To solve this problem the following change is proposed:

$$Q(s,a;\theta) = V(s;\theta) + A(s,a;\theta) - \frac{1}{|\boldsymbol{A}|}\sum_{a' \in \boldsymbol{A}} A(s,a';\theta) \tag{7}$$

where the mean value is subtracted from all values, so that the advantage always sums to zero.

4 Tasks

This work is divided into two parts. In the first part your main goal is to implement a Convolutional Neural Network (CNN) that takes as input an RGB image (or a stack of RGB images) corresponding to the visual representation of the OpenAI **CarRacing-v2** environment. Since this is a hard scenario, and considering time constrains in the Google Colab, the goal condition is to obtain a total reward per episode higher than 200 in the previous 20 episodes (in one of the control modes), however keep in mind that it is possible to obtain a value close to 900 using DQN, Double DQN or Dueling DQN, depending on the network and hyperparameters selected. By default, the **CarRacing-v2** generates a new car track per episode, take this into account when analyzing your results. You must implement DQN, Double DQN and Double Dueling DQN, and train the models in the environment's discrete and continuous modes (see the **UseContinuousControl** flag), to that end, you must complete the following tasks:

- Prepare a CNN and modify the necessary hyperparameters in the provided Google Colab notebook;
- Implement the Double DQN and Dueling DQN in the same Google Colab notebook, use the available flags to switch between methods (Double Dueling DQN is also a valid method);
- Train each model in continuous and discrete modes, compare the results between each model and each control modality.
- Evaluate the best models in 5 random tracks and in 5 random environments (see the **UseRandomize** flag);
- Experiment with Q-learning hyperparameters (Deep layers, action replay memory, batch size, gamma, epsilon, episode, etc). The provided Google Colab notebook lets you define the number of consecutive frames used to represent the state and the number of iterations the same action is executed in the environment.

Note: If for some reason you are unable to achieve the minimum results, you can force the environment to use the same seed and always generate the same track, which will make the environment easier.

For the second part, you must select based on the performance achieved in the first part which method to use. This part is composed of two scenarios, one scenario without obstacles (other than the ones at the borders) and a scenario with obstacles. For the first scenario, Modes.PoseOnly should be used, while for the second scenario Modes.PoseOnly and Modes.PoseAndLocalMapInline can be used. The code is made available in one of the colab files related to the assignment. Test scenarios are defined in the colab file, but changes can be made to increase or decrease the scenario's difficulty. Similarly, the number of actions or the reward model can be modified.

5 Deliverables

Each group (two students) must deliver the implemented **Google Colab notebook (.ipynb)** and a detailed report (**cannot exceed 15 pages**) with:

- **Part I**:
  - Diagram of the implemented network architectures used to solve the **CarRacing-v2** OpenAI environment;
  - A description of the implemented architectures;
  - Any additional implementation details, as well as, any diagrams you see fit to highlight the different stages of your implementation. Include all the hyperparameters used;
  - Compare the results obtained, report which method has the highest performance and justify;
  - Discuss the effects of the Deep Q-learning hyperparameters on the reported performances;
  - Always include the rewards per episode and the evaluation in 10 scenarios (5 with random tracks but static scenarios and 5 with random tracks and random scenarios) when reporting the performance of each method.
  - Select the best performing method (from any control modality) and **retrain** it using **UseRandomize=True**, discuss the obtained results;
- **Part II**:
  - Diagram of the implemented network architectures used in the custom environment;
  - Test the models with different goal and starting positions, in both modes, comment the results;
  - Discuss the results obtained and which changes (if any) should be introduced to improve model performance in the scenarios.
  - All changes to the environment's code should be clearly documented in the report.

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] H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with Double Q-Learning. In *AAAI*, 2016.

[3] Ziyu Wang, Tom Schaul, Matteo Hessel, H. V. Hasselt, Marc Lanctot, and N. D. Freitas. Dueling network architectures for deep reinforcement learning. *ArXiv*, abs/1511.06581, 2016.

[4] Christopher J. C. H. Watkins and Peter Dayan. Technical Note: Q -Learning. *Machine Learning*, 1992.

[5] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to Deep Reinforcement Learning. *CoRR*, 2018.

[6] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *ArXiv*, abs/1810.12282, 2018.

[7] Hado Hasselt. Double Q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.