1 Introduction

In this assignment it is expected that you learn and understand how to apply Deep Reinforcement Learning (DeepRL) techniques in controlling agents in simulation environments. This assignment is split in two parts: affordance-based state representation and 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 two simulation environments from the OpenAI gym will be explored, namely **CartPole-v0** and **CarRacing-v0**. In the **CartPole-v0** environment, a pole is attached to a cart which moves along a frictionless track. The pole starts upright and the goal is to prevent it from falling over by controlling the cart's velocity. The environment provides 4 observations (that will comprise the DeepRL state representation): the cart's position, the cart's velocity, the pole's angle, and the pole's angular velocity. To control the cart two actions are available: push the cart to the left and push the cart to the right. The reward model is constant, meaning that it is 1 for every step including the termination state. The environment default conditions to terminate an episode are: pole angle is more than 12 degrees, cart position reaches the edge of the display, and episode length is greater than 200.

The **CarRacing-v0** environment is a harder challenge where a car must race trough a track. The RL state representation consists in a DeepRL state composed of a 96×96 RGB image. To control the car 3 different variables can be modified: steering wheel, gas, and break (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 break corresponds to the break pedal (range: [0,1]). While using DQN or DQN-based methods, the action values must be discretized into a set of suitable actions. The goal track is randomly generated at every episode. 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.

3 Affordance-based state representation

3.1 Tasks

- Your main goal is to model a Deep Neural Network that takes as inputs sensor observations corresponding to the OpenAI **CartPole-v0** environment. The goal condition is to obtain a total reward per episode higher than 195 in the previous 20 episodes. To accomplish you can:
  - use epsilon-greedy for exploration;
  - explore DQN, Double DQN and Double Dueling DQN (comparison with all methods will be valued);
  - use a priority replay buffer;
  - add/remove fully-connected layers;
  - add/remove dropout or add other regularization methods;
  - modify the activation functions;
  - change the loss function and/or change the optimization method;
  - change any hyper-parameter;
  - use an adaptive learning rate.

4 Image-based state representation

4.1 Tasks

- Your main goal is to model a Convolutional Neural Network that takes as input an RGB image corresponding to the visual representation of the OpenAI **CarRacing-v0** 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, however keep in mind that it is possible to obtain a value close to 900 using a DQN, depending on the network and hyper-parameters selected. By default the **CarRacing-v0** generates a new car track per episode, however to avoid problems due to hardware constrains the code only considers the first track generated. To accomplish this you can:
  - use epsilon-greedy for exploration;
  - explore DQN, Double DQN and Double Dueling DQN (comparison with all methods will be valued);
  - modify the action set;
  - adapt the reward models;
  - use a priority replay buffer;
  - add/remove fully-connected layers;
  - add/remove dropout or add other regularization methods;
  - modify the activation functions;
  - change the loss function and/or change the optimization method;
  - change any hyper-parameter;
  - use an adaptive learning rate;
  - start by adapting the CNN from the first assignment;
  - use a pretrained model (e.g., VGG16), however you should not freeze its weights;
  - add/remove convolutional layers;
  - add max-pooling layers.

5 Deliverables

- A detailed report with:
  - diagram of the implemented network architectures used to solve the two OpenAI environments;
  - a description of the implemented CNN architecture;
  - results for four hyper-parameter combinations;
  - you should write any additional implementation details, as well as, any diagrams you see fit to highlight the different stages of your implementation (e.g., data loading, training, validation, evaluation);
  - your report should also answer the following questions:
    - Compare the performance of the Deep Q-Learning architecture in both state representations. Compare also the training time. Assess the drawbacks of each representation from the observed performances.
    - Propose a new reward model (if necessary) for the **CartPole-v0** environment. You do not need to implement the new reward model but should clearly compare and highlight the advantages and disadvantages of both models. If you think there is no need for a new model you should explain your reasoning.

    - Compare the performance of the Deep Q-Learning architecture with DQN, Double DQN and Double Dueling DQN. How can you explain the gap in performance (if any)?
    - Experiment with Q-learning hyperparameters (Deep layers, action replay memory, batch size, gamma, epsilon, episode, etc). Discuss the effect of the Deep Q-learning hyperparameters on the methods performance. You should find a parameter that makes a nontrivial difference on the performance. Training for a larger number of episodes would likely yield further improvements in performance? Explain your answer.
    - Run the best model obtained in the OpenAI **CarRacing-v0** environment using only one track in an unmodified version (one track per episode). Comment the results.
  - **Extra:** Adapt the code used in the **CarRacing-v0** environment to run in the **Pong-v0** environment and comment the results.
- Google Colab notebooks (ipynb)