Project 1: Navigation

The project demonstrates the ability of value-based methods, specifically, Deep Q-learning and its variants, to learn a suitable policy in a model-free Reinforcement Learning setting using a Unity environment, which consists of a continuous state space of 37 dimensions, with the goal to navigate around and collect yellow bananas (reward: +1) while avoiding blue bananas (reward: -1). There are 4 actions to choose from: move left, move right, move forward and move backward.

The following report is written in four parts:

- **Implementation**
- **Results**
- **Ideas for improvement**

Implementation

For this project I used a Deep Q-Network (DQN) model. The aim is to train a policy that tries to maximize the discounted, cumulative reward Rt0, where Rt0 is also known as the return. I used PyTorch for the implementation of the DNN. The experience gained by the agent while acting in the environment is saved in a memory buffer, and a small batch of observations from this list is randomly selected and used as the input to train the weights of the DNN (i.e., Experience Replay). At each step, the agent follows an greedy approach to select an action. The value of is continuously decayed after every episode to gradually focus on exploitation rather than exploration.

Hyperparameters

There were many hyperparameters involved in the experiment. The value of each of them is given below:

Hyperparameter	Value
Replay buffer size	1e5
Batch size	64
γ (discount factor)	0.99
τ	1e-3
Learning rate	5e-4
update interval	4
Number of episodes	500
Max number of timesteps per episode	2000
Epsilon start	1.0
Epsilon minimum	0.1
Epsilon decay	0.995

Results:

The best performance was achieved by **Double DQN** where the reward of +13 was achieved in **377** episodes. It was a bit confusing to see that Dueling Double DQN wasn't the best one but I attribute it to the fact that I didn't do a hyperparameter search for the same and instead used the same hyperparameter setting. The plots of the rewards for the different variants of

DQN is shown below:

Ideas for improvement

- Using Prioritized Replay (paper) showed a massive improvement over Double DQNs for Atari games. It is expected that it'll lead to an improved performance here too.
- Other improvements to the original DQN algorithms that were briefly mentioned in the course could be potentially beneficial too: learning from multi-step bootstrap targets , Distributional DQN, Noisy DQN
- Hyperparameter search for both Double DQNs and Dueling Double DQNs should lead to better performance too.