Navigating a Collector Agent

1 Introduction

Reinforcement learning is based on the study of an agent interacting with an environment to maximize some notion of cumulative reward. Where the environment provides the state, the reward and the learner (agent) has to select an action based on a policy Q.

In this project I trained an agent using DQN (Deep Q-Learning), this algorithm uses a deep neural network represented action value function q*. For this, the DQN does not need any additional knowledge about the task in order to learn. It takes the state as input and produces probabilities of performing every possible action in that state. However, when neural networks are used to represent the value-action function, reinforcement learning becomes unstable. So we use two techniques to stabilize.

- Experience replay: after each time interval, the agent accumulates a tuple of state, action, reward and next state that after it is discarded, what the experience buffer does is store this tuple to use in learning;
- Fixed Q-targets: The algorithm uses a target function to represent the old Q function;

2 Method

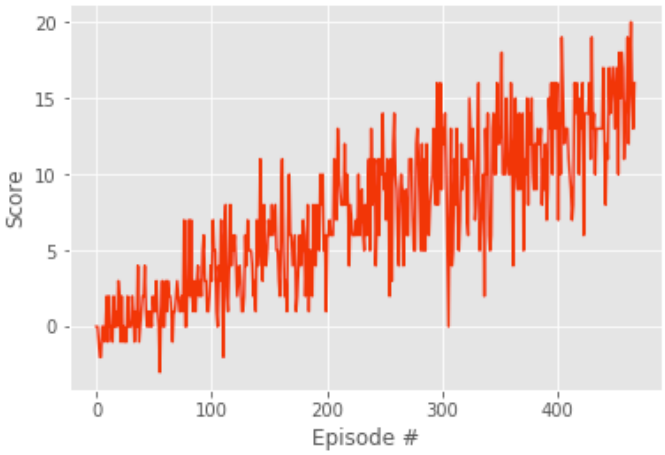
The Q-value function implemented to solve this problem is based on Q-networks, so the network architecture used is a simple and fully connected neural network with 2 hidden layers of sizes, 64 units. After each fully connected layer, a rectified linear function (ReLU) is applied, except after the output layers. For stability reasons, gradients are cut during training and passed backwards. Weights are randomly initialized and experience samples selected from the repeat buffer to provide more information.

The purpose of this network is to learn the approximate function that translates 37 state values to 4 action values. Therefore, the hyperparameters used are:

- The buffer size is set to 100,000 so that all experiments can be added to a maximum memory size limit.
- The batch size is kept at 64.
- The range is kept at 0.99, so the discount factor for future rewards is 0.99.
- Tau is kept at 1e-3, being an update parameter to the destination network.
- The learning rate is 5e-4 which controls weight updates.
- Refresh every 4 time steps, experiments are sampled from memory and the target network is updated.
- Epsilon starts at 1 starting by exploring the environment and decays at a rate of 0.995.

3 Results

Training starts with an e-greedy policy that exploits the environment most likely known as a greedy policy while decreases exponentially from 1.0 to 0.01. This random policy is used for filling the replay buffer with at least 5,000 sample experiments. The stopping condition is reached once, as well as training, as the validation results pass 13 points as a score. The training results are shown in figure.



4 Improvements

In this project, DQN (Deep Q-Learning) was implemented, but there are improvements that can be made to the algorithm for faster training with better accuracy.

- Double DQN where it uses two function approximators to overestimate the action values, one to select and one to estimate the effectiveness of the action.
- Dueling DQN that evaluates the value of each state whether it is valuable or not.
- Prioritized Repetition prioritizes important experiments by placing a higher sample probability.