Andrea Pena

Udacity DRLND

Project 1: Navigation

Description of the Project

Note: To resolve this project (P1: Navigation), I used a similar architecture that I used in the DQN coding exercise in this lesson.

The goal of the navigation project is training an agent to navigate (and collect bananas!) in a large, square world. A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Therefore, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas. The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent must learn how to best select actions.

Four discrete actions are available, corresponding to:

- 0 - walk forward
- 1 - walk backward
- 2 - turn left
- 3 - turn right

The task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

Learning algorithm

As I stated before, this project utilized a Deep Q-Learning. This algorithm continues episodical training via a DQN agent until n_episodes are reached or until the environment is solved. The environment is considered solved when the average reward (over the last 100 episodes) is at least +13. Each episode continues until max_t time-steps is reached or until the environment says it's done. A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. The DQN agent is enclosed in a file named “dqn_agent.py”. For each time step the dqn_state acts on the current state and epsilon-greedy values. The dqn_agent utilize a replay buffer of experiences.

DQN Hyper Parameters

- n_episodes (int): maximum number of training epissodes
- max_t (int): maximum number of timesteps per episode
- eps_start (float): starting value of epsilon, for epsilon-greedy action selection
- eps_end (float): minimum value of epsilon
- eps_decay (float): multiplicative factor (per episode) for decreasing epsilon

In this case, the environment was solved in 206 episodes. To achieve that, I used these final hyper-parameters used were as follows: n_episodes=600, eps_start=1.0, eps_decay=0.98, eps_end=0.02

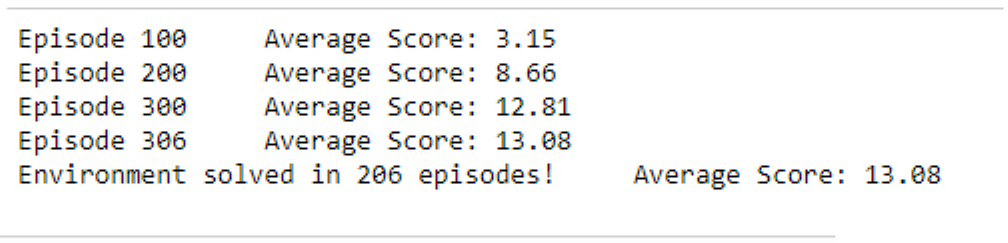
DQN Agent Hyper Parameters

- BUFFER_SIZE (int): replay buffer size
- BATCH_SIZ (int): mini batch size
- GAMMA (float): discount factor
- TAU (float): for soft update of target parameters
- LR (float): learning rate for optimizer
- UPDATE_EVERY (int): how often to update the network

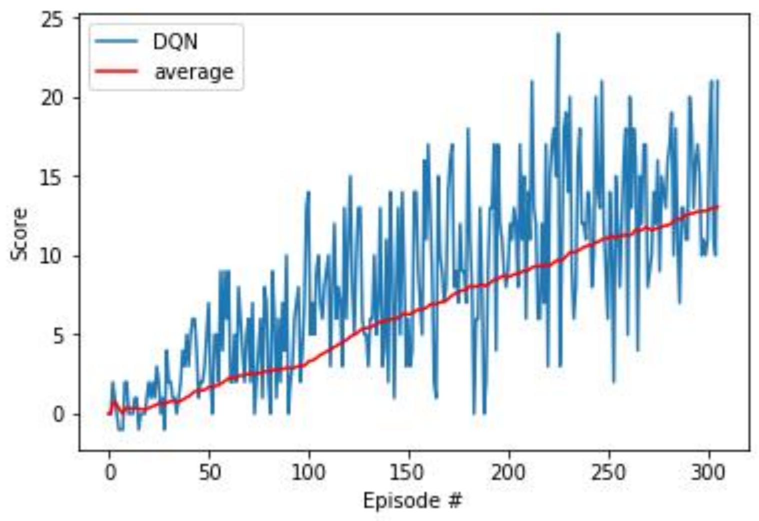
In this project, I used the following values: BUFFER_SIZE = int(1e5), BATCH_SIZE = 64, GAMMA = 0.99, TAU = 1e-3, LR = 5e-4 and UPDATE_EVERY = 4

Neural Network: In this project, the QNetwork use 2 x 64 fully connected layers with Relu activation and a final fully connected layer with the same number of units as the action size. In addition, the QNetwork has the same size that the state size.

Plot of Rewards



```
Episode 100     Average Score: 3.15
Episode 200     Average Score: 8.66
Episode 300     Average Score: 12.81
Episode 306     Average Score: 13.08
Environment solved in 206 episodes!     Average Score: 13.08
```



Future Work

I would recommend for future improvement to prioritize experience replay instead of the random selection of tuples. In that way, the prioritized will select experiences based on a priority value which is correlated with the magnitude error. In that manner, it will improve the learning by increasing the probability of the experience vectors that are sampled.

Furthermore, another improvement will be to have a way to enable and disable the replay buffer. Because in this project all agent used the replay buffer and the test results don't measure the impact the replay buffer has on performance.