Project One: Navigation

1. Project Overview

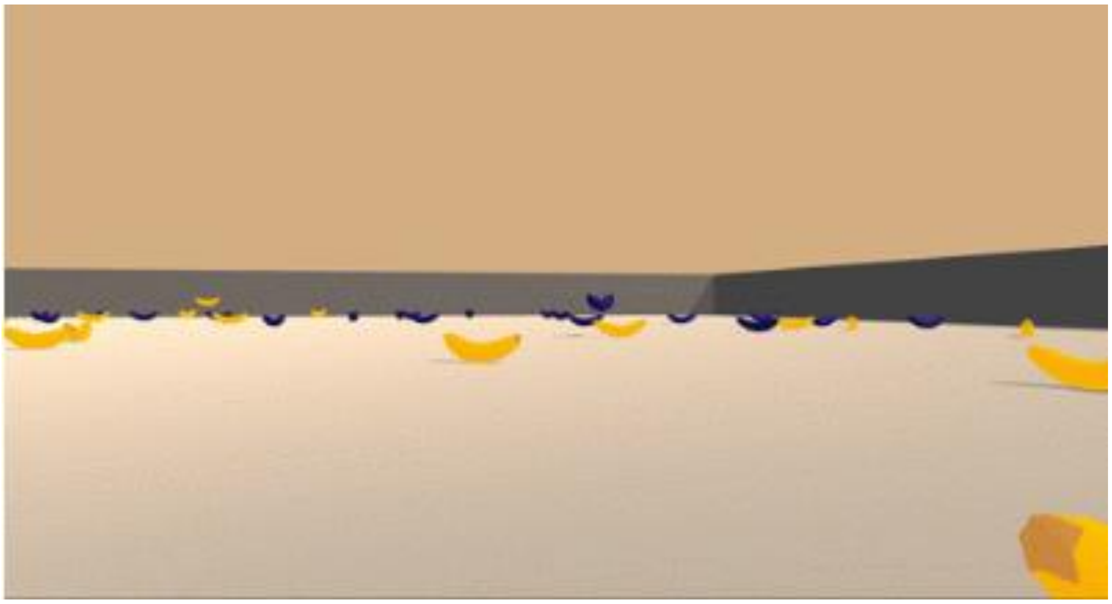
In this project, I trained an agent to navigate (and collect bananas!) in a large, square world.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, 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 has to learn how to best select actions. Four discrete actions are available, corresponding to:

- 0 - move forward.
- 1 - move backward.
- 2 - turn left.
- 3 - turn right.

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



2. Learning Algorithm

This project used the Deep Q-Networks Algorithm. I applied a DQN implementation to the ML-Agents toolkit on Windows10 operating system. And I tweak the various hyperparameters and settings to build my intuition for what should work well (and what doesn't!).

In Deep Q-Networks Algorithm, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards rt discounted by c at each timestep t, achievable by a behaviour policy π = P (a | s), after making an observation (s) and taking an action (a).

The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

in which c is the discount factor determining the agent’s horizon, hi are the parameters of the Q-network at iteration i and ϑi are the network parameters used to compute the target at iteration i. The target network parameters ϑi are only updated with the Q-network parameters (ϑi) every C steps and are held fixed between individual updates.

The following figure shows the Schematic illustration of the convolutional neural network.

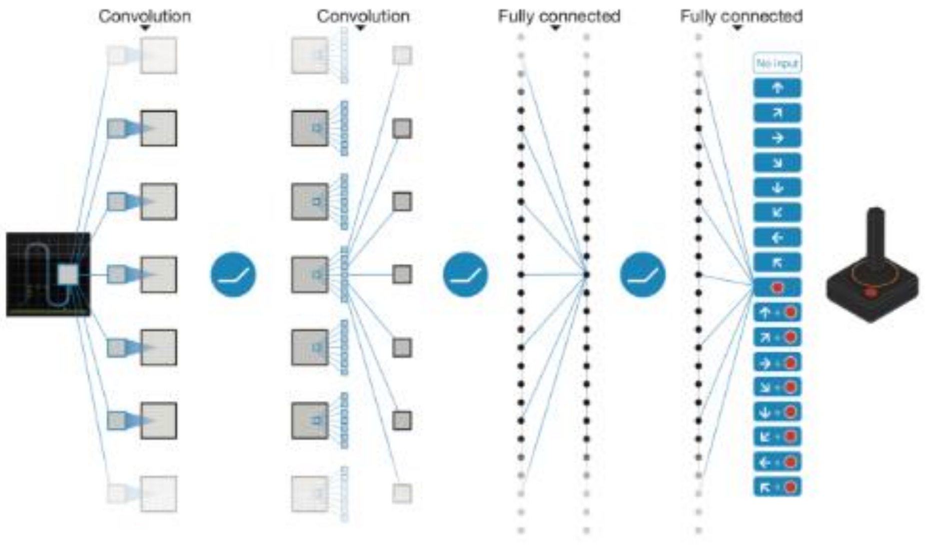


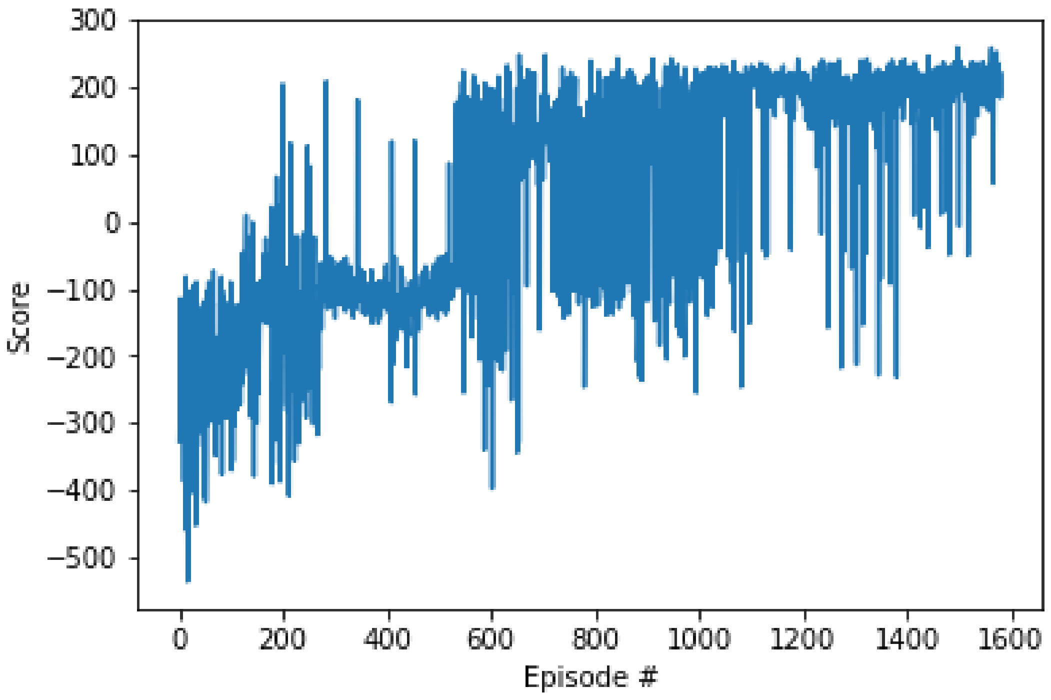
Illustration of DQN Architecture (Source)

3. Plot of Rewards

In my implementation for the project by the Deep Q-Networks Algorithm, to optimize the hyper-parameters, choosing parameter ‘eps_decay’ as: eps_decay=0.9925

Here, ‘eps_decay’ is a multiplicative factor (per episode) for decreasing epsilon.

By running the Python code for the implementation, an output plot as following shows the relationship between episode and score.



And the training's output results are as following:

```
Episode 100     Average Score: -217.75
Episode 200     Average Score: -156.64
Episode 300     Average Score: -133.95
Episode 400     Average Score: -103.04
Episode 500     Average Score: -115.09
Episode 600     Average Score: -17.049
Episode 700     Average Score: 63.727
Episode 800     Average Score: 30.73
Episode 900     Average Score: 30.90
Episode 1000    Average Score: 59.96
Episode 1100    Average Score: 134.44
Episode 1200    Average Score: 193.28
Episode 1300    Average Score: 169.51
Episode 1400    Average Score: 169.59
Episode 1500    Average Score: 187.92
Episode 1582    Average Score: 201.24
Environment solved in 1482 episodes!  Average Score: 201.24
```

From the outputs above, we can see that the average reward over 100 episodes is 13.579 (201.24/14.82). And solved the environment in only 1482 episodes. This is satisfied by project requests.

4. Ideas for Future Work

We could also use the double DQN algorithm, or a dueling DQN algorithm to implement this project. Double Q-Learning has been shown to work well in practice to help with this. By replacing the traditional Deep Q-Network (DQN) architecture with a dueling architecture, we can assess the value of each state, without having to learn the effect of each action.