DQN for Banana collection game environment

Report

I. Environment

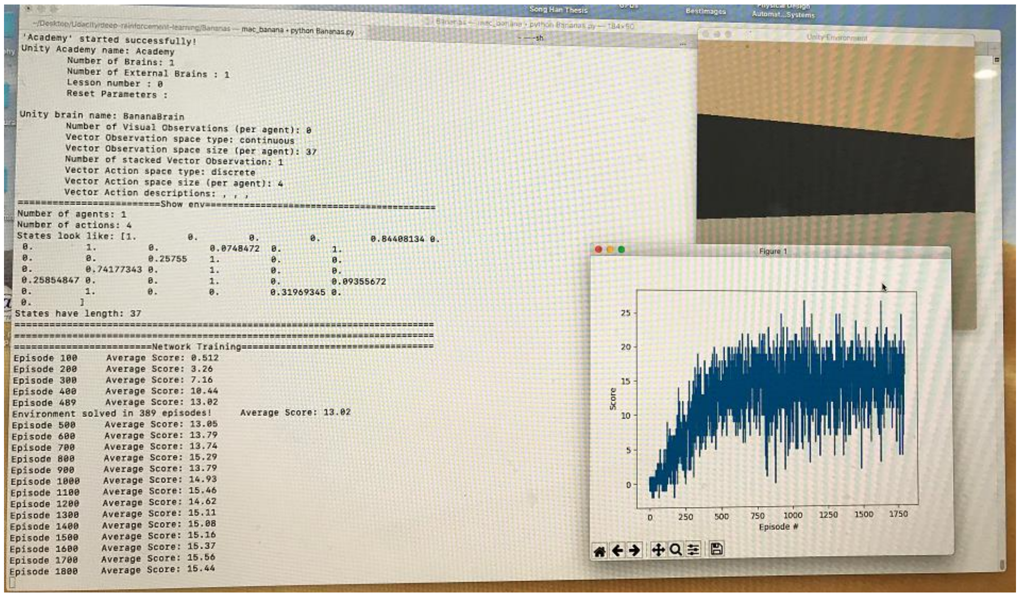
A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas. The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent must 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, the agent must get an average score of +13 over 100 consecutive episodes.

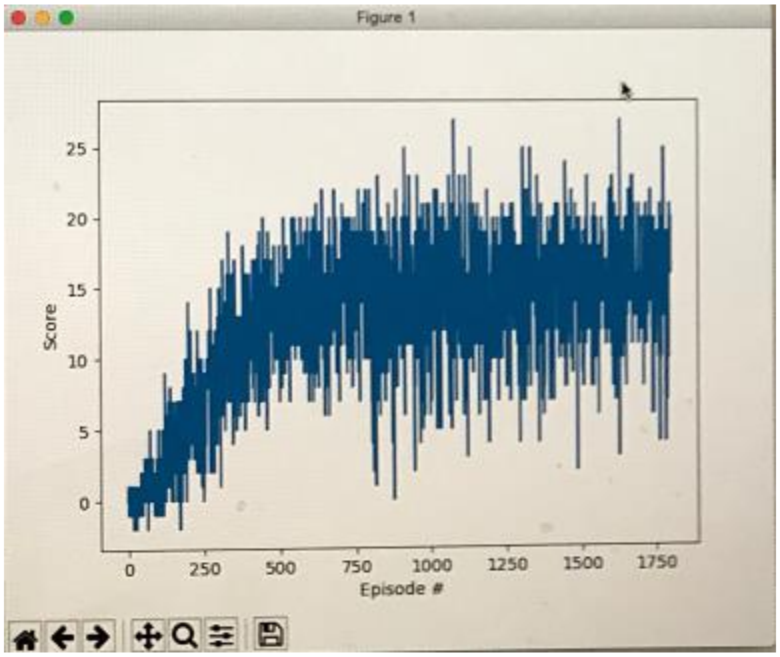
II. Results

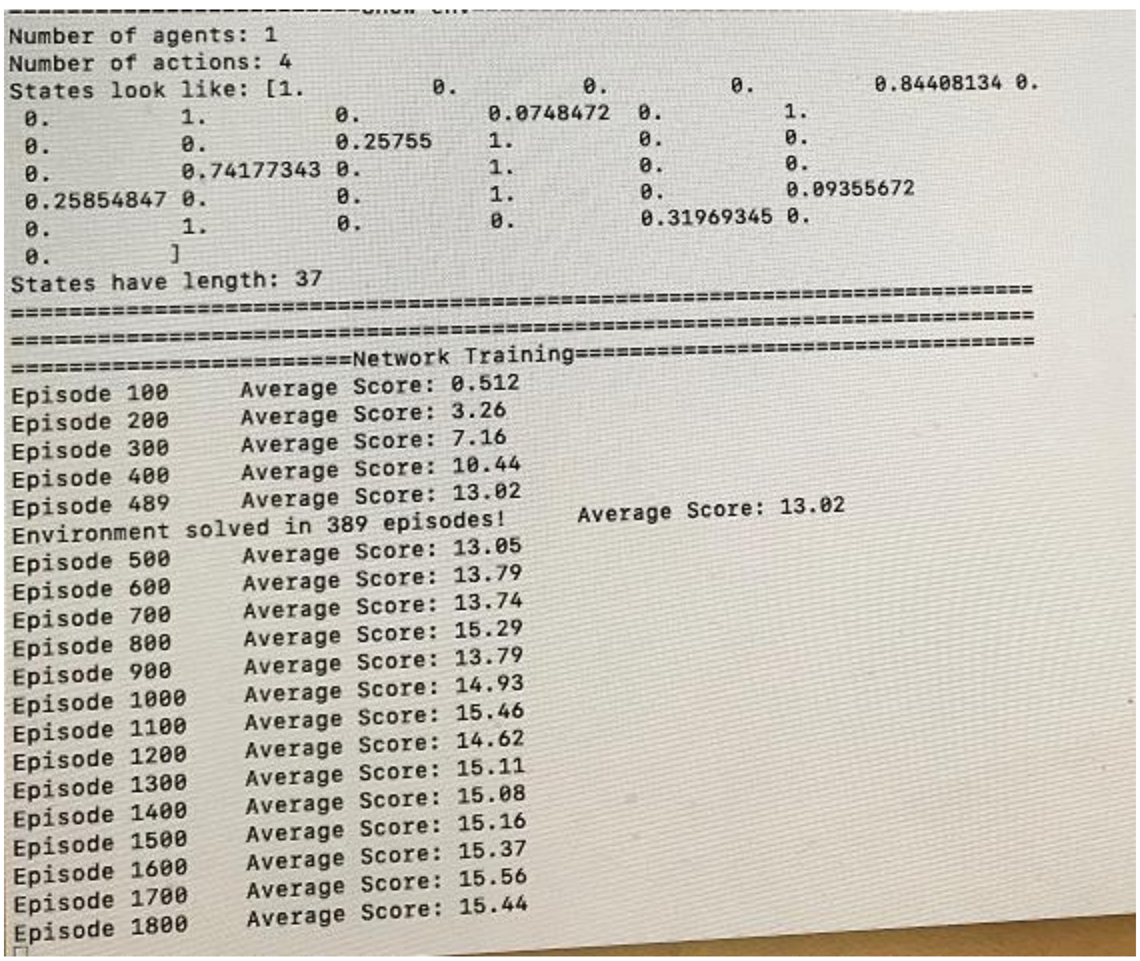
The DQN network was an MLP with 2 hidden layers. The input layer has 37 neurons as there were 37 observations from the environment and the output layer has 4 neurons. The network achieved an average of 13+ over 100 consecutive episodes within 600 iterations of training. The training report screen shots are below.


~/Desktop/Udacity/deep-reinforcement-learning/Bananas — mac_banana • python Bananas.py
Figure 1
Score
Episode #
0
250
500
750
1000
1250
1500
1750
0
5
10
15
20
25

```
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :

Unity brain name: BananaBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 37
        Number of stacked Vector Observation: 1
        Vector Action space type: discrete
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
========================Show env========================================
Number of agents: 1
Number of actions: 4
States look like: [1.         0.         0.         0.         0.84408134 0.
 0.         1.         0.         0.0748472  0.         1.
 0.         0.         0.25755    1.         0.         0.
 0.         0.74177343 0.         1.         0.         0.
 0.25854847 0.         0.         1.         0.         0.09355672
 0.         1.         0.         0.         0.31969345 0.
 0.        ]
States have length: 37
=======================================================================
=======================================================================
========================Network Training================================
Episode 100     Average Score: 0.512
Episode 200     Average Score: 3.26
Episode 300     Average Score: 7.16
Episode 400     Average Score: 10.44
Episode 489     Average Score: 13.02
Environment solved in 389 episodes!     Average Score: 13.02
Episode 500     Average Score: 13.05
Episode 600     Average Score: 13.79
Episode 700     Average Score: 13.74
Episode 800     Average Score: 15.29
Episode 900     Average Score: 13.79
Episode 1000    Average Score: 14.93
Episode 1100    Average Score: 15.46
Episode 1200    Average Score: 14.62
Episode 1300    Average Score: 15.11
Episode 1400    Average Score: 15.08
Episode 1500    Average Score: 15.16
Episode 1600    Average Score: 15.37
Episode 1700    Average Score: 15.56
Episode 1800    Average Score: 15.44
```





```
Number of agents: 1
Number of actions: 4
States look like: [1.         0.         0.         0.         0.84408134 0.
 0.         1.         0.         0.0748472  0.         1.
 0.         0.         0.25755    1.         0.         0.
 0.         0.74177343 0.         1.         0.         0.
 0.25854847 0.         0.         1.         0.         0.09355672
 0.         1.         0.         0.         0.31969345 0.
 0.        ]
States have length: 37
==========================================================================
==========================================================================
========================Network Training==================================
Episode 100     Average Score: 0.512
Episode 200     Average Score: 3.26
Episode 300     Average Score: 7.16
Episode 400     Average Score: 10.44
Episode 489     Average Score: 13.02
Environment solved in 389 episodes!     Average Score: 13.02
Episode 500     Average Score: 13.05
Episode 600     Average Score: 13.79
Episode 700     Average Score: 13.74
Episode 800     Average Score: 15.29
Episode 900     Average Score: 13.79
Episode 1000    Average Score: 14.93
Episode 1100    Average Score: 15.46
Episode 1200    Average Score: 14.62
Episode 1300    Average Score: 15.11
Episode 1400    Average Score: 15.08
Episode 1500    Average Score: 15.16
Episode 1600    Average Score: 15.37
Episode 1700    Average Score: 15.56
Episode 1800    Average Score: 15.44
```

III. Future Improvements

There are several improvements to the original algorithm that is discussed in literature a few examples that are well suited for simple incremental improvements are discussed below.

1. Prioritized Experience Replay
   Prioritized experience replay is based on the idea that the agent can learn more effectively from some episodes of the game is of more value than other. So a priority is assigned to data in the replay buffer for retrieval and learning.

   The central component of prioritized replay is the criterion by which the priority of each transition is measured. One criterion would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable proxy is the magnitude of a transition's TD error δ, which indicates how 'surprising' or unexpected the transition is: specifically, how far the value is from its next-step bootstrap estimate.

   The algorithm stores the last encountered TD error along with each transition in the replay memory. The transition with the largest absolute TD error is replayed from the memory. A Q-learning update is applied to this transition, which updates the weights in proportion to the TD error. New transitions arrive without a known TD-error, so we put them at maximal priority in order to guarantee that all experience is seen at least once.

   Since there are thousands of transitions an optimal data structure is needed to store and retrieve those with higher priority. A heap implemented on an array can be used to get samples with highest priority faster.

   So tan improvement would be to add TD error estimated along with transitions, make the queue a priority queue and make sampling guided by priority.

2. Double DQN.
   This improves estimation of action value. The max operator in standard Q-learning and DQN, uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this, Double DQN decouples the selection from the evaluation. This is the idea behind Double Q-learning