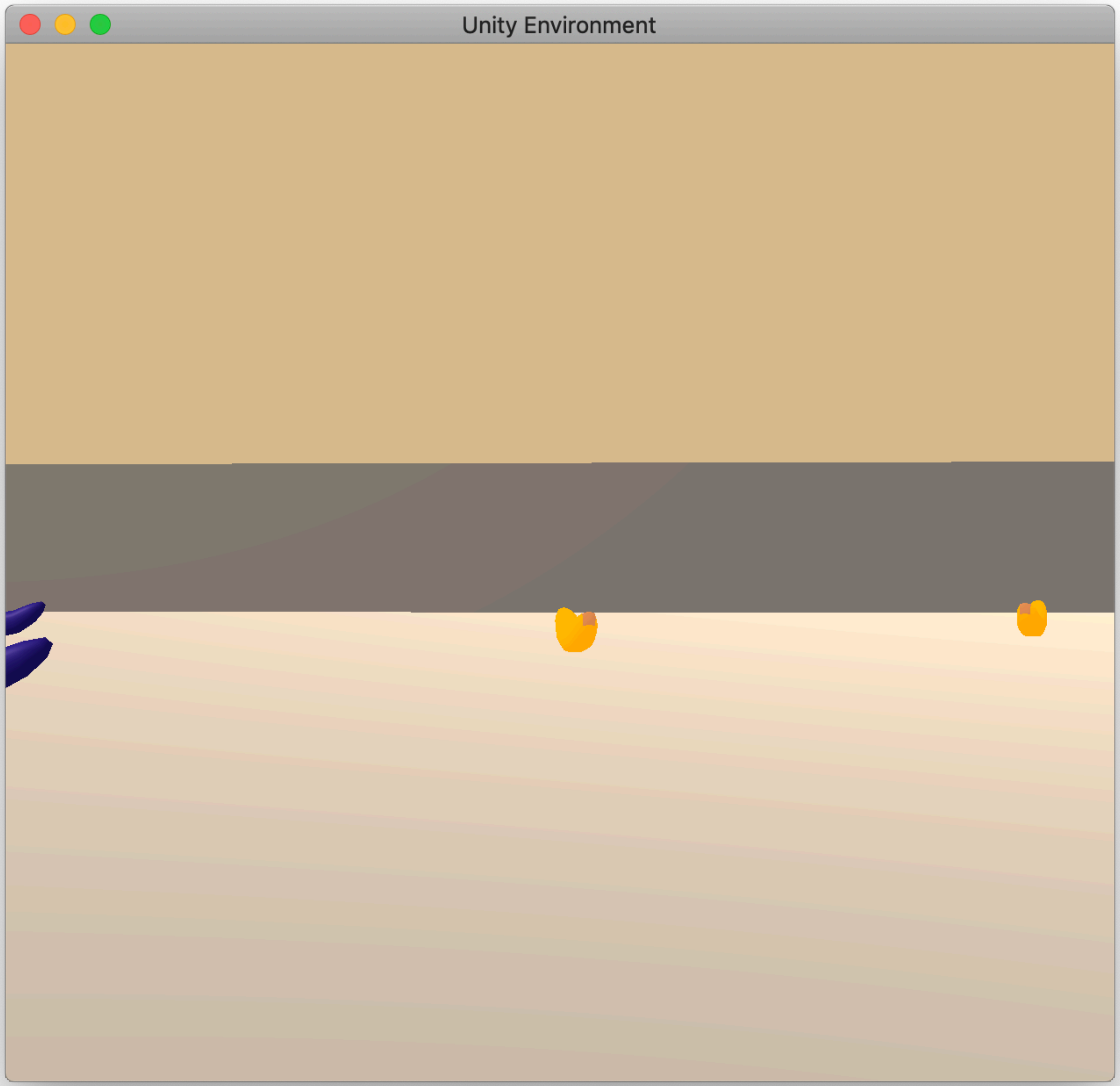
Ali Abdullah Al-Saeedi
Reinforcement Nanodegree
Navigation Project
March 25, 2019

Navigation



This project is about 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. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas. The task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

1.Implementation

This project is based on *Deep Q-Learning* which is an off policy which means the policy being evaluated is different from the one being learned.

Q-learning in general is to learn the action-value function, denoted as Q(s , a) , where “s” is the current state and “a” is the action being evaluated. It is based on the following equation:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \bigg)}^{\text{learned value}}$$

[1]

Since our space is continuous we have to use *Function Approximator* to have an approximation of the action-value function. Here we use the Deep Learning as the *Function Approximator* and thats why it is called *Deep Q-Learning*. We use the following equation to update the weights:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w}) \big] \nabla \hat{q}(S, A, \mathbf{w})$$

[1]

In this implementation we used what is called *Double DQN* for the model

- **Double DQN**: The popular Q-learning algorithm is known to overestimate action values under certain conditions. It was not previously known whether, in practice, such overestimations are common, whether they harm performance, and whether they can generally be prevented. The recent DQN algorithm, which combines Q-learning with a deep neural network, suffers from substantial overestimations in some games in the Atari 2600 domain. The idea behind the Double Q-learning algorithm, which was introduced in a tabular setting, can be generalized to work with large-scale function approximation [2].

Hyperparameters:

There are many Hyperparameters that can be edited:

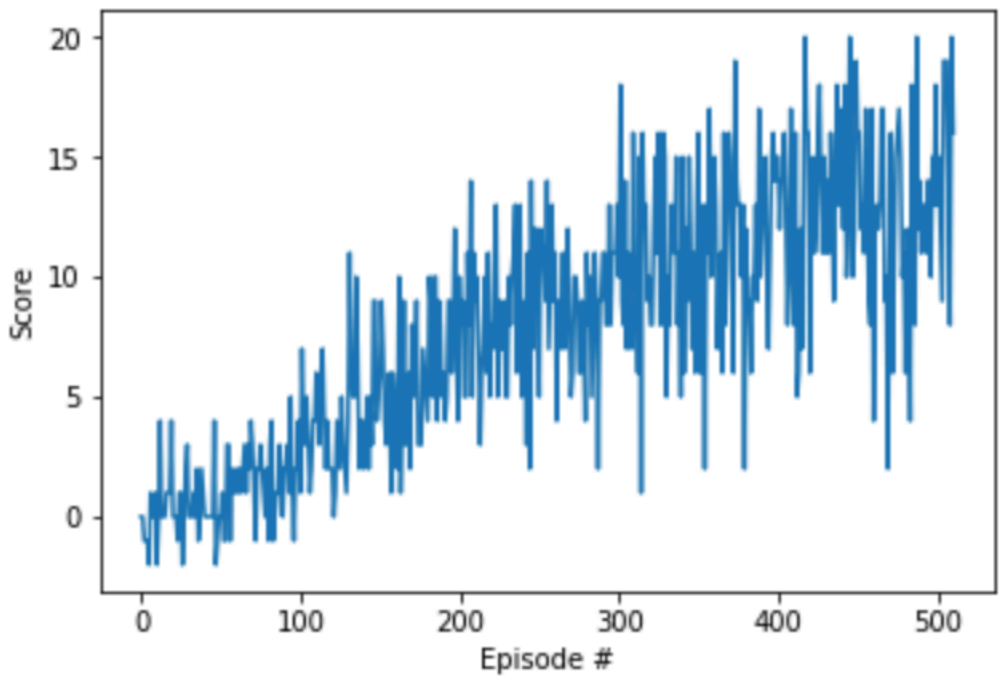
Number of episodes	n_episodes=2000
Max time per episodes	max_t=1000 (ms)
Epsilon start	eps_start=1.0
Epsilon minimum	eps_end=0.01
Epsilon decay	eps_decay=0.995

2.Results

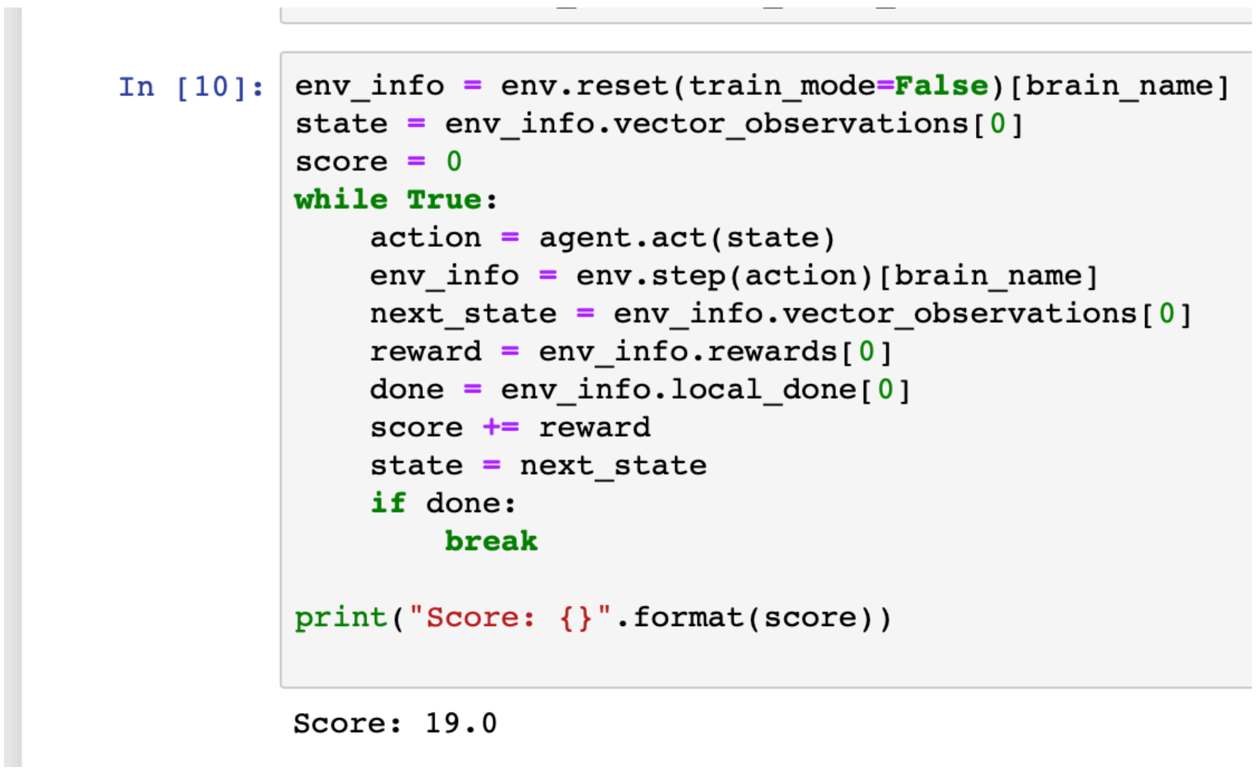
In the training we could solve the environment in 411 episodes. By solving it we mean reaching an average score of at least 13.

```
Episode 100	Average Score: 0.93
Episode 200	Average Score: 4.99
Episode 300	Average Score: 8.64
Episode 400	Average Score: 11.04
Episode 500	Average Score: 12.74
Episode 511	Average Score: 13.01
Environment solved in 411 episodes!	Average Score: 13.01
```

This graph shows how while our agent is training we are gaining a better result.



Finally let’s test our trained agent and try it in a game.



```
In [10]: env_info = env.reset(train_mode=False)[brain_name]
         state = env_info.vector_observations[0]
         score = 0
         while True:
             action = agent.act(state)
             env_info = env.step(action)[brain_name]
             next_state = env_info.vector_observations[0]
             reward = env_info.rewards[0]
             done = env_info.local_done[0]
             score += reward
             state = next_state
             if done:
                 break

         print("Score: {}".format(score))

         Score: 19.0
```

We can see that our agent could win the game successfully and got a result of 19.

1.Future Work

- We could use different algorithms to try if our agent would give us a better results on training. Such algorithms would be Dueling Network.
- Using techniques like Prioritized Replay would give a great improvement over Double DQNs .

1.Sources

1.Wikipedia

2.Deep Reinforcement Learning with Double Q-learning