Project: Navigation

Name: Zain Us Sami Ahmed Ansari

Introduction

For 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 our 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 agent's forward direction. Given this information, the agent has to learn how to best select actions.

Implementation

In this exercise the parameters that are tuned are as follows.

```
BUFFER_SIZE = int(1e4)  # replay buffer size
BATCH_SIZE = 32         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 1e-3               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

The model architecture is defined through set of following variables

state_size (37) which represents input to the network. action_size (4) which represents output of the network, seed is used to initialise the weights of the network.

The parameters that define the network are as follows

```
"""Initialize parameters and build model.
  Params
  ======
      state_size (37)(int): Dimension of each state
      action_size (4)(int): Dimension of each action
      seed (int): Random seed
      fc1_units (64)(int): Number of nodes in first hidden layer
      fc2_units (64)(int): Number of nodes in second hidden layer
  """
```

The DQN is trained using the following parameters.

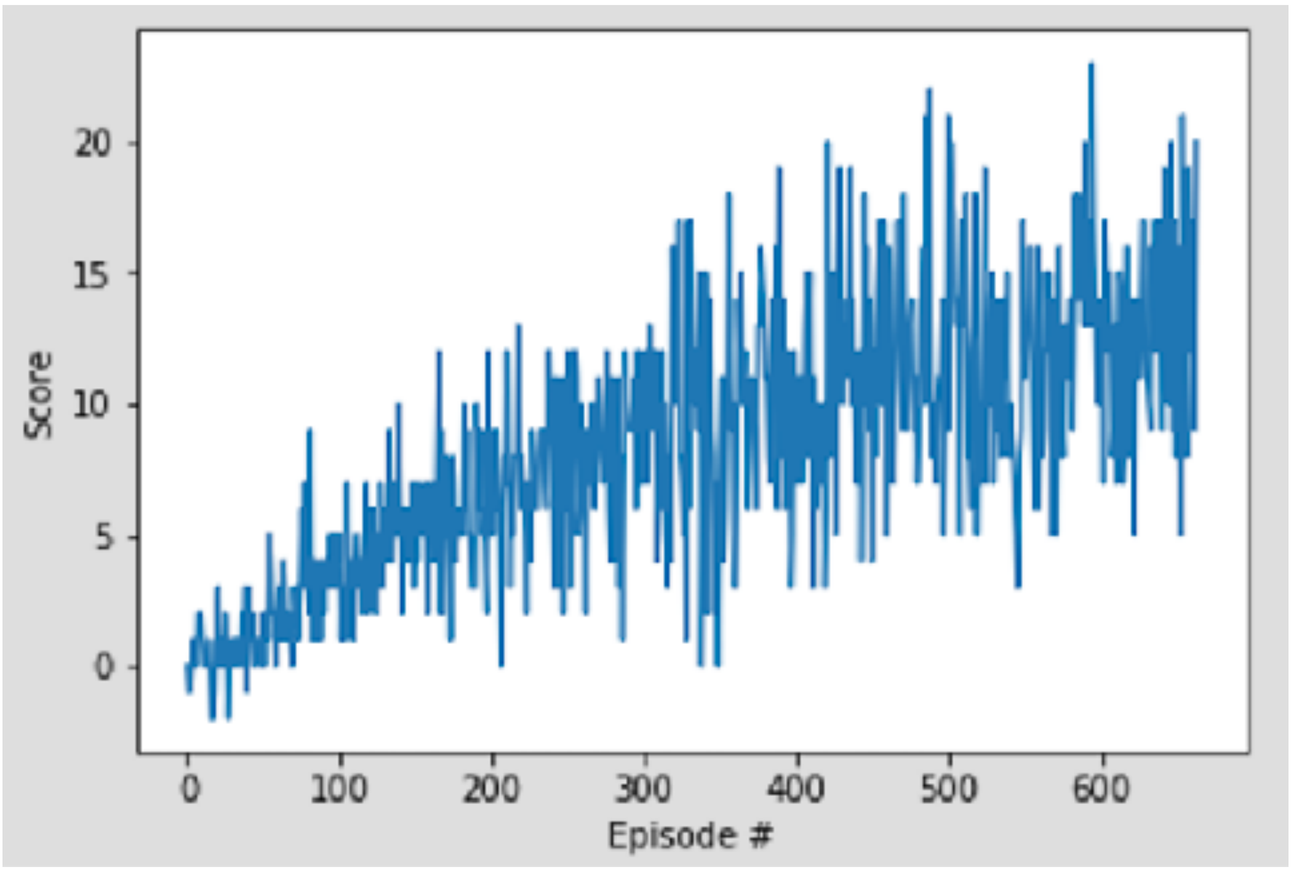
```
 Params
    ======
        n_episodes (900)(int): maximum number of training episodes
        max_t (1000)(int): maximum number of timesteps per episode
        eps_start (1.0)(float): starting value of epsilon, for epsilon-greedy action selection
        eps_end (0.01)(float): minimum value of epsilon
        eps_decay (0.01)(float): multiplicative factor (per episode) for decreasing epsilon
    “""
```

The Deep Q Network samples the network as a non linear function to calculate the value for actions based on the environment observations.

To reduce oscillations in the neural network the implementation uses experience replay.

In the learn step a small batch of experiences are randomly sampled to train the agent.

Plot of Rewards



Future Work

As it is very evident from the plot of rewards that the standard deviation of this implementation is very high I want to try out different parameters and network architectures to reduce the standard deviation.

As mentioned in the project suggestions I would also try Double DQN, prioritised experience replay and duelling DQN to improve episode performance and increase stability.