Project Name: Navigation

Introduction

As part of Navigation project, we leveraged deep reinforcement learning to help train Unity-ML agent pick the TASTY YELLOW BANANAS with increasing accuracy instead of ROTTEN BLUE BANANAS. I leveraged DQN (Deep Q Network) as described in the paper
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Neural Network Architecture

As part of the Q network architecture we have built for this project, we were given 37 inputs (state space) and leveraged 2 hidden layers as well as 1 output layer with 4 outputs/classes (0- move forward, 1 - move backward, 2- turn left, 3 - turn right) to build final solution for this project . Thus, our final neural network looks as follows:

1. Input Layer - 37 inputs
2. Hidden Layer 1 - Fully connected layer
3. Hidden Layer 2 - Fully connected layer
4. Output Layer - Full Connected with 4 classes

Under the Hood

As part of this implementation, complete code / python notebook can be found at location:
https://github.com/divya-bhanot/Project1_Navigation/blob/master/Navigation.ipynb

Hyper parameters used

Hyper Parameters are parameters which are given by the users to train a model. Similarly, for DQN we defined several hyper-parameter to ensure we get optimized results.
Though it took me some iterations to come up with right set of values, following are the final values which helped me get optimum threshold results and their significance:

- Buffer Size -> This hyper-parameter define how many values of state, action, etc. we could store in memory. For our model, we have defined this to be 1e5 (100,000)

- Batch Size -> Number of examples to be picked up when the agent updates the learning curve i.e. for learning updates

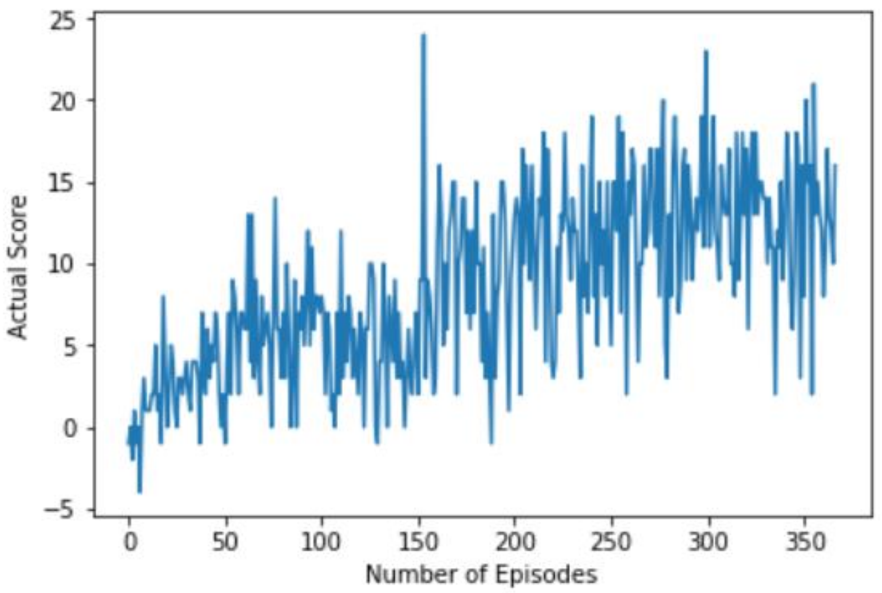
- Gamma -> The discount factor which we have used is multiplied by future rewards in order to dampen these rewards' effect on the agent's choice of action. We have used discount value of 0.99.

- Learning Rate -> This rate denotes how the learning curve optimizes the results in each run and minimize the loss function.

- Update Frequency -> It implies how many steps the agent is taking for each Q-learning iteration.

Output

Below is the plot of the rewards per episode experience by the agent during final solution training



Future Changes

Here are list of improvements which we could do to improve the performance of learning

1. Better Algorithms -> Instead of using Standard DQN algorithm, we could use Double DQN algorithm

2. Hyper Parameter optimization -> We could leverage more combinations of different hyper parameter, to ensure better performance and faster convergence. Though I did
Tried many combinations, there was still scope of more hit and trials to come up with more optimal value using a different set of values of Hyper parameter