Deep reinforced learning Project one

Introduction:

In this project, we are going to train our agent to collect as many yellow banana as we can without collect the blue banana. We are using the Deep Q learning approach.

Learning Algorithms:
The learning algorithms we are going to use is the deep Q network. This is different from the classic Q learning where we simply replace the Q table with the "Function Approximator", which is a deep neural network.

The file model.py contains the actual deep neural network which is using the pytorch library and framework. The dqn_agent.py file use the model and implement the deep Q network that support the training and loading the trained model to be executed. There are two classes defined in the dqn_agent.py file that implemented the ReplayBuffer class and the Agent class.

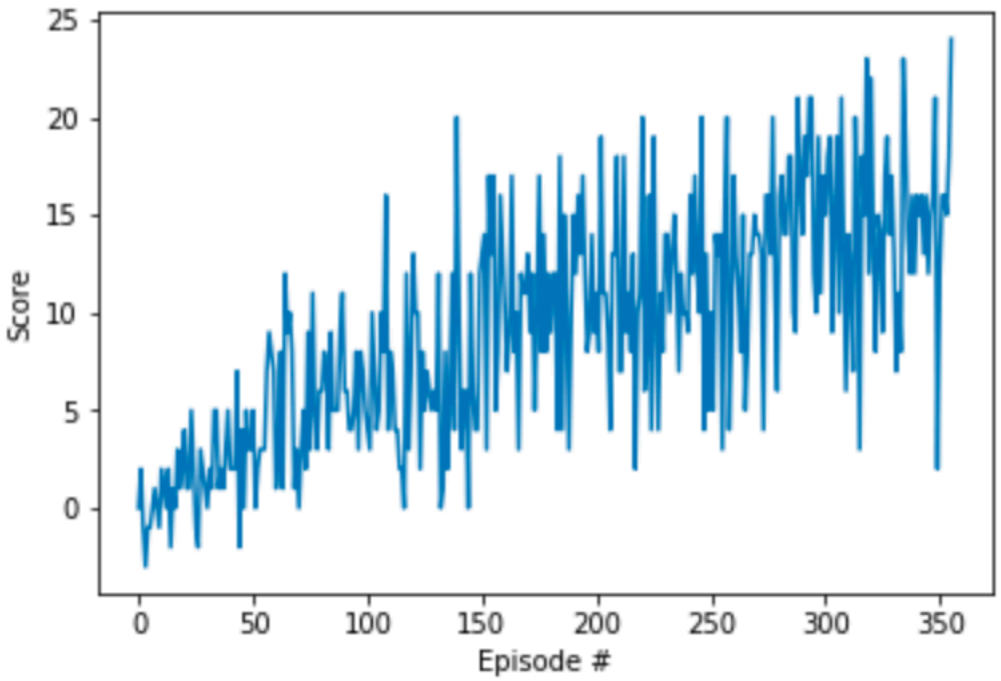
The ReplayBuffer class is responsible to save or store the previous experience for later to be used to train the network so that the Agent get a chance to explore base on pass experienced. This is the class we use as Replay Memory in the actual DQN. By sampling from it randomly, the transitions that build up a batch are decorrelated. Research shown that this greatly stabilize the improves the DQN training procedure.

The Agent contains the DQN algorithm. The Epsilon-Greedy Policy is used where we start our epsilon from 1 and decay 0.999 every single episode until it reach 0.01. In each episode, we perform 1000 steps and browse the banana picking world and calculate the reward and the next state. There are total of 37 states and each state contains 4 actions.

Again, the deep neural network is used as Function Approximator and we are using the 3 layers' network model with 64 hidden units in each layer and the input size of 37 dimensions and the output are 4 possible, which represent the 4 possible for each state. We put this information back to our environment so that we know our next state.

Training Result:

Training tool slightly less than 10 minutes on an iMac with Intel i5 processor (without GPU support for Pytorch). It took slightly less than 360 episodes.



Possible future improvement:

The current implement can achieve reasonably good result. However, we should always look for improvement. Without using different algorithms, we can only do two things to improve. Either fine-tune the hyper-parameter further or use different neural network. I believe fine-tuning the hyper-parameter may not be able to achieve a significant amount of improvement except for training time. Whereas, increasing the complexity of the neural network may be able to help achieve a greater result.
Standard DQN suffers from overestimate of Q value in early stage while it is still evolving. Double Q-Learning mitigates the issue by having separate set of weights to evaluate the action. To improve it even further, Dueling DQN may be used for accessing the value of the state without having to evaluate the value of each action.