Deep Reinforcement Learning Nanodegree

Project #1: Navigation

Implementation

Brief description: This coding project implements Deep Q-Learning to navigate a large, flat world, while collecting yellow bananas and avoiding blue bananas scattered throughout the environment.

Model architecture: The base QNetworks are defined in *model.py*, with 3 fully connected layers, with the first and second layers both having 64 nodes and the final layer having 4 nodes (i.e. an action vector). Rectified linear unit (**ReLU)** activation functions are used between the two hidden layers. These are used as approximators for the optimal action-value function.

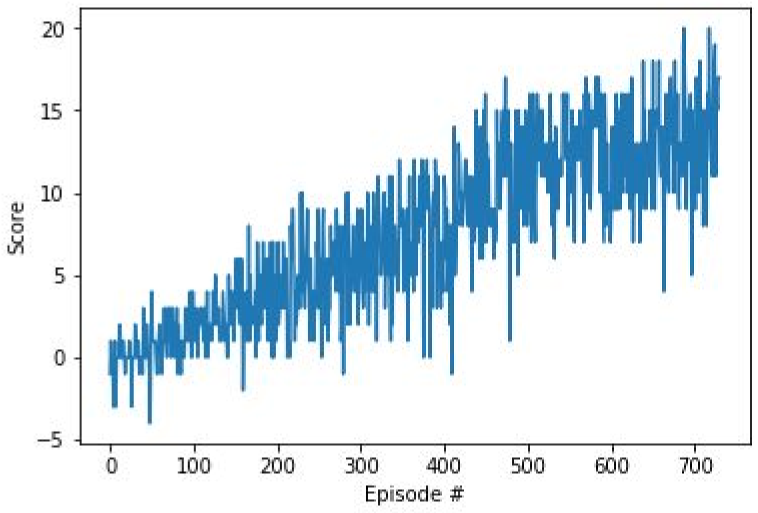
Deep Q-Learning: The method used for this project is known as Fixed-Q Targets, wherein two models with the same architecture are instantiated, with one being designated as the local network and the other as the target network. The target network is updated only once certain criteria are met, that is, that the current local network has sampled a sufficient amount of state-action pairs from the environment and 4 time-steps have passed. This is to alleviate the issue of "chasing a moving target" that arises from using only one network. The optimizer used is the Adam optimizer (a variant of stochastic gradient descent) whose goal is to minimize the mean squared error between the target network's predicted action value for the subsequent states and the local network's predicted action value for the current states. The target network is updated using a soft update so it is not fully overwritten, the amount of the target network which is preserved is determined by the hyperparameter tau. The implementation can be seen in *model.py*.

Hyperparameters:
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.99
TAU = 1e-3
LR = 5e-4
UPDATE_EVERY = 4
eps_decay = 0.997

final_eps = 0.01
eps_start = 1.0

Score Per Epoch Over Training

```
Environment solved in 630 episodes.     Average Score Attained: 13.06
Training took 942.2542090415955 seconds.
```



Future improvements:

- Additional tweaks to the model could include utilizing a dueling Q-network, double Q-network and prioritized experience replay.
- Instead of using observations from the environment, try to use image data directly.
- Further hyperparameter tuning and model architecture optimization.