Navigation Project Report

Learning Algorithm

This model used a Deep Q-Learning algorithm with the following parameters:

Replay buffer size: int(1e5)
Minibatch size: 64
Discount factor: 0.99
For soft update of target parameters: 1e-3
Learning rate: 5e-4
How often to update the network: 4
Optimizer: Adam

Two identical networks are created – one “local” and one “target”. Every time the agent takes an action and receives a reward, the original state, action, reward, and resulting state are recorded in a buffer, as well as whether the episode is done or not.

In this network, every 4 steps the network checks if there is enough information in the buffer to learn from (ie. if the length of the buffer is greater than the minibatch size), and if so, then it takes a random subset and learns from it.

“Learning” in this case refers to getting the max predicted Q values for the next states from the target model, and comparing them to the expected Q values of the local model. The loss is then calculated and the local model is updated and then copied over to the target model.

The architecture for the local and target models includes 4 linear layers: input, output and two hidden layers. A relu function is used between layers but not on the output layer.

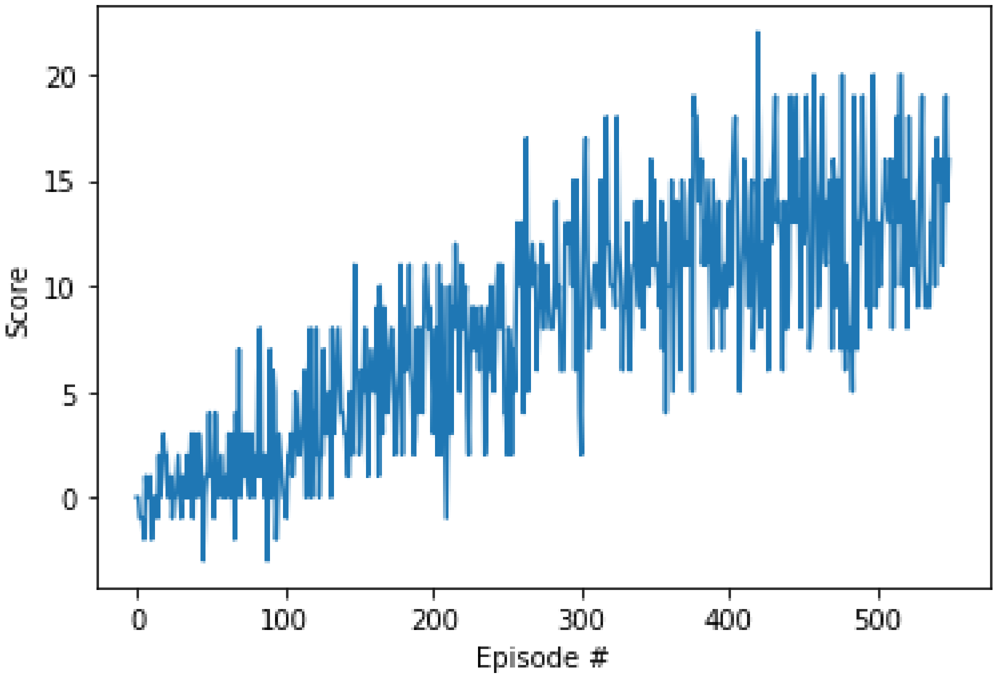
According to openai.com, a DQN algorithm such as the one used in this project is a “reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments, like video games, or robotics.” (https://openai.com/blog/openai-baselines-dqn/)

In a reinforcement learning environment, an agent is placed into a new environment of which it has no previous knowledge. It then tries to determine which actions are best to take in the environment that will lead to it receiving the highest reward.

A DQN algorithm works well for this environment because there are clear states and actions and a good reward system is set up (+1 for every yellow banana and -1 for every blue banana). The agent uses trial and error to determine which actions bring it the highest reward.

Plot of Rewards

```
Episode 100	Average Score: 1.10
Episode 200	Average Score: 4.77
Episode 300	Average Score: 8.17
Episode 400	Average Score: 11.28
Episode 500	Average Score: 12.70
Episode 549	Average Score: 13.01
Environment solved in 449 episodes!	Average Score: 13.01
```



Ideas for Future Work

This agent may perform better with the Double DQN, Dueling DQN and/or Prioritized Experience Replay improvements to the Deep Q-Learning algorithm. I would be especially interested in trying out the Rainbow algorithm to solve this environment.