Project: Navigation

The main aim of this is to train an agent to learn by itself to navigate (and collect specific bananas!) in a large, square world. The goal is to maximize the rewards by collecting a yellow banana and avoiding blue bananas in a give number of episodes. Given that the agent must learn to select the best actions, a deep reinforcement network is used.

For this project, an environment built in Unity is used.

Learning Algorithm

To solve the environment, a Deep Q Network (DQN) is used. Deep Q-Learning algorithm represents the optimal action-value function q_* as a neural network (instead of a table).

Some of the parameters used include:

- `BUFFER_SIZE = 100000: This replays the buffer size`
- `BATCH_SIZE = 64: This is the size of each minibatch`
- `GAMMA = 0.99: This is the discount factor`
- `TAU = 0.001: This is used for soft update of target parameters`
- `LR = 0.0005: This is the learning rate`
- `UPDATE_EVERY = 4: This represents how often to update the network`
- `eps_start = 1.0`
- `eps_end = 0.01`
- `eps_decay = 0.995`
- `num_episodes = 1800: This is the number of episodes the agent is to train for`

The replay buffer contains a collection of experience tuples (S, A, R, S′). Based on the parameters selected for the network, 100,000 of these experience tuples are stored at a time. Small batch of tuples from the replay buffer is sampled in order to learn. This allows for the agent to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of the experience.

The agent uses the epsilon-greedy policy to select the next action. From this, the agent will initially favor exploration by randomly selecting an action from the action space with a probability of epsilon. As the agent learns some more, the epsilon gradually decays and thus favoring exploitation over exploration.

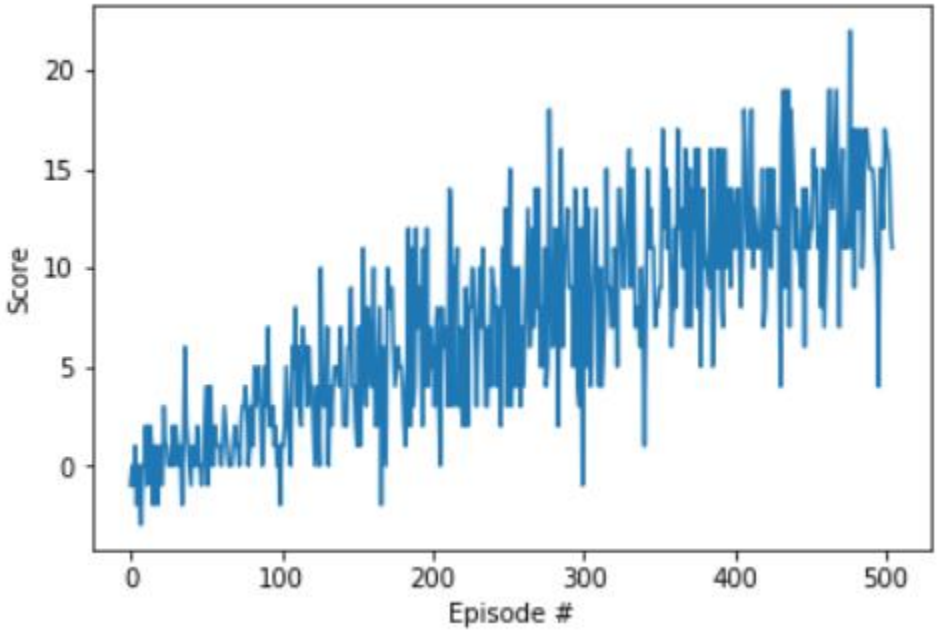
Model Architecture

To solve the environment, three fully connected linear layers with ReLU activation function is used.

```
self.fc1 = nn.Linear(state_size, 64)

self.fc2 = nn.Linear(64, 64)

self.fc3 = nn.Linear(64, action_size)

x = F.relu(self.fc1(state))

x = F.relu(self.fc2(x))

x = self.fc3(x)
```

Result

From the requirement of the project, the environment is considered solved when the agent gets an average score of +13 over 100 consecutive episodes.



Based on the requirement for the project, the environment was solved in 405 episodes!

Ideas for Future Work

To improve the performance of the agent, I am plan on implementing the Double DQN and also the dueling DQN.