Navigation Project Report

Learning Algorithm

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

Replay buffer size: int(1e5)
Minibatch size: 64
Discount factor = 0.99
For soft update of target parameters = 1e-3 # for soft update of target parameters
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.

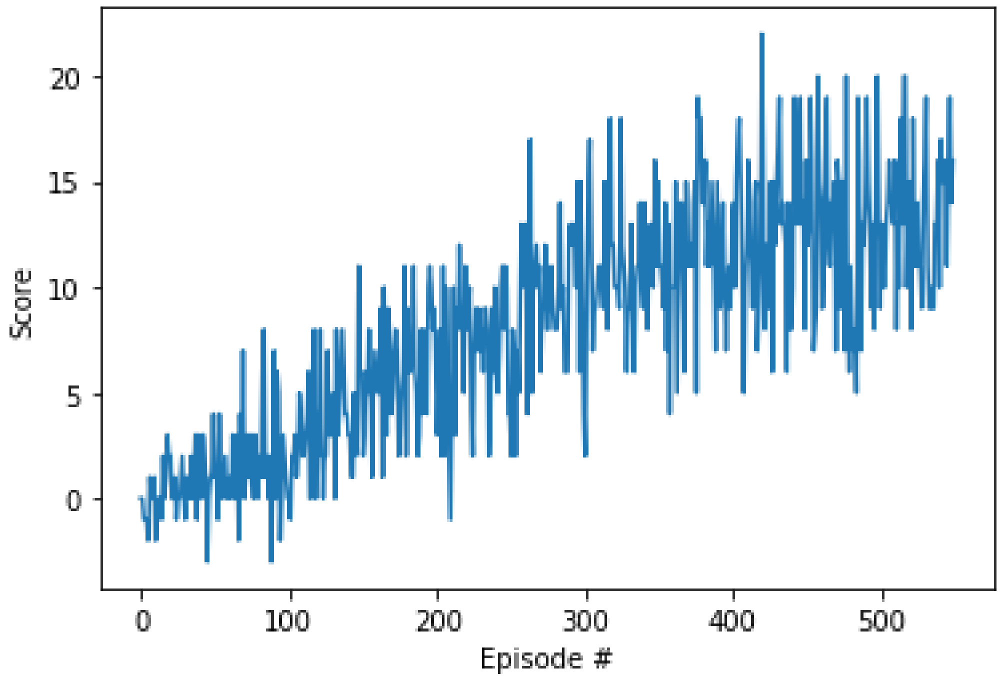
In this network, every 4 steps the network checks if there is enough information in the buffer to learn from (ie. if the length of the buffer is greater than the minibatch size), and if so, then it takes a random subset and learns from it.

"Learning" in this case refers to the 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 & target models include 4 linear layers: input, output and two hidden layers. A relu function is used between layers put not on the output layer.

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 network. I would be especially interested in trying out the Rainbow algorithm to solve this environment.