DQN Agent for Navigation:

Learning Algorithm:

I have solved the Banana Navigation environment by building and training a Deep Q Network agent.

Q Network Model Architecture:

Feedforward network with 2 hidden layers. Each hidden layer has 64 units/neurons. Hidden layers have used ReLu activation function.

DQN Agent:

Agent interacts with the environment using epsilon greedy policy and learns using Adam optimizer. DQN agent comprises the Q neural network defined above which holds the learnable parameters.

This agent is capable of taking action in the environment using Epsilon-greedy policy. Once the agent takes an action in a given state - environment returns reward and the next_state for that time step. This experience tuple of (state, action, reward, next_state) is saved in Memory/Replay Buffer.

These experiences are then sampled from the Memory for the learning step - where next_state is first used to calculate Q_target_next and then Q_target_next is used to compute Q_targets. While the action actually taken and current state is used to calculate Q_expected.

Once both Q_targets and Q_expected are computed - these 2 values are used to compute loss and then this loss is used by the optimizer to update the neural network weights.

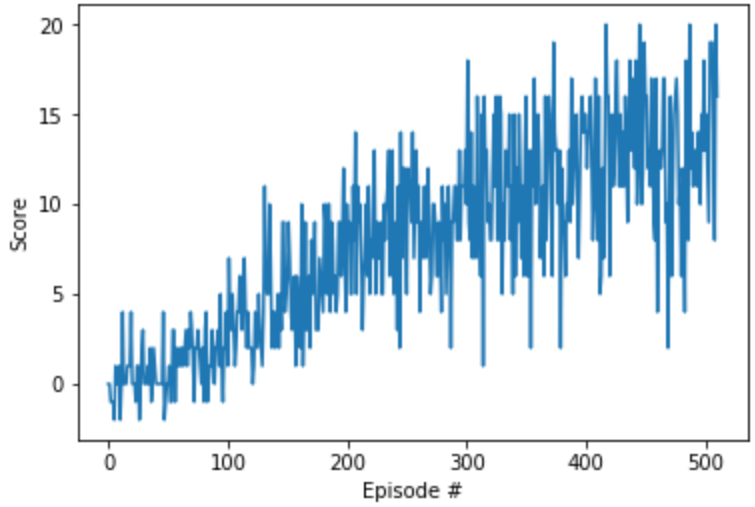
Also - every 4 steps - model weights are copied from the local_model and are updated to target_model as part of a soft_updare process. Below are the chosen hyperparameters:
BUFFER_SIZE = int(1e5) # replay buffer size
BATCH_SIZE = 64   # minibatch size
GAMMA = 0.99   # discount factor
TAU = 1e-3   # for soft update of target parameters
LR = 5e-4   # learning rate
UPDATE_EVERY = 4   # how often to update the network

Training:

Agent is trained in episodes and there are steps in each episode. Below are the chosen parameters for training:
n_episodes=2000, max_t=1000, eps_start=1.0, eps_end=0.01, eps_decay=0.995

Plot of Rewards:

Environment solved in 411 Episodes! Plot of rewards below.



Ideas for future Work:

A vanilla DQN agent is used to learn to solve the Navigation environment. In future we can explore using Double DQN, Prioritized Experience Replay, Dueling DQN and the RAINBOW architecture to improve the performance of the RL agent further.

Hyperparameter search methods like Bayesian optimization can also be used to further fine tune hyperparamaters which can also improve RL agent's performance.