Deep Reinforcement Learning Nanodegree Project 1 Report

Ohn Kim

Learning Algorithm

I used DQN algorithm to solve this problem.

Deep Q-Network was first introduced in DeepMind's "Playing Atari with Deep Reinforcement Learning," which developed AlphaGo. State-action value Q is approximated through deep learning. This model uses the agent state as an input and outputs the Q action value.

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
- n_episodes = 2000 # maximum number of training episodes
- max_t = 1000 # maximum number of time steps per episode
- eps_start = 1.0 # starting value of epsilon, for epsilon-greedy action selection
- eps_end = 0.01 # minimum value of epsilon
- eps_decay = 0.995 # multiplicative factor (per episode) for decreasing epsilon

Model architecture

The model is made of five fully connected layers. At the end of each fc layer, a ReLU activate function was applied. (Not in the last fc layer.) The layers have different numbers of neurons. The figure below shows how many neurons each layer has.

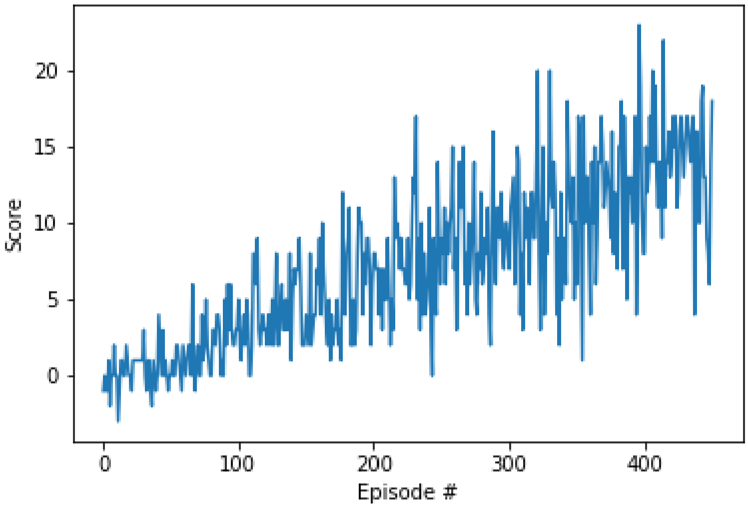
```
self.fc1 = nn.Linear(state_size, 256)
self.fc2 = nn.Linear(256, 128)
self.fc3 = nn.Linear(128, 64)
self.fc4 = nn.Linear(64, 32)
self.fc5 = nn.Linear(32, action_size)
```

of episodes needed to solve the environment

```
Episode 100     Average Score: 1.08
Episode 200     Average Score: 4.68
Episode 300     Average Score: 7.98
Episode 400     Average Score: 10.93
Episode 451     Average Score: 13.01
Environment solved in 451 episodes!     Average score: 13.01
```

After a total of 451 episodes, average score is over +13.

Plot of rewards



Ideas for Future Work

I used the DQN model to solve this problem. This model could have solved the problem enough, but other algorithms could be used for better performance. Using DQN's various extensions (Dueling DQN, Noisy DQN, Rainbow model, etc) could achieve better results.