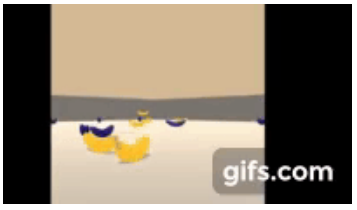
Report for Project 1 - Deep Reinforcement Learning Nanodegree



This report contains the details of the code used to solve the first project in Udacity's Deep Reinforcement Learning Nanodegree program.

The algorithm

The agent uses a policy to move in the environment and find the optimal policy to maximize its reward. The optimal policy is the result of a trial and error effort through which the agent leans by iteration. This kind of algorithm is called Q-Learning and the optimal policy is found by running the Q-function which calculates the expected reward for all possible actions and all possible states.

The agent uses a deep neural network to approximate the Q-function. The Deep Q-Network contains three fully connected layers of 64, 64, 4 nodes. The learning rate is set to: 0.0006 Decreasing the learning rate has given better results during training.

The agent uses also an Experience Replay, the experience is stored in a buffer and the agent samples randomly form this buffer during the learning steps.

The experiments and hyperparameters

I have tried different learning rates and found out an interesting thing: the agent learns from previous trainings as well! By starting the agent with the trained weights and decreasing the decay, the training results even more effective and the training speeds up, resolving the environment in very few episodes!

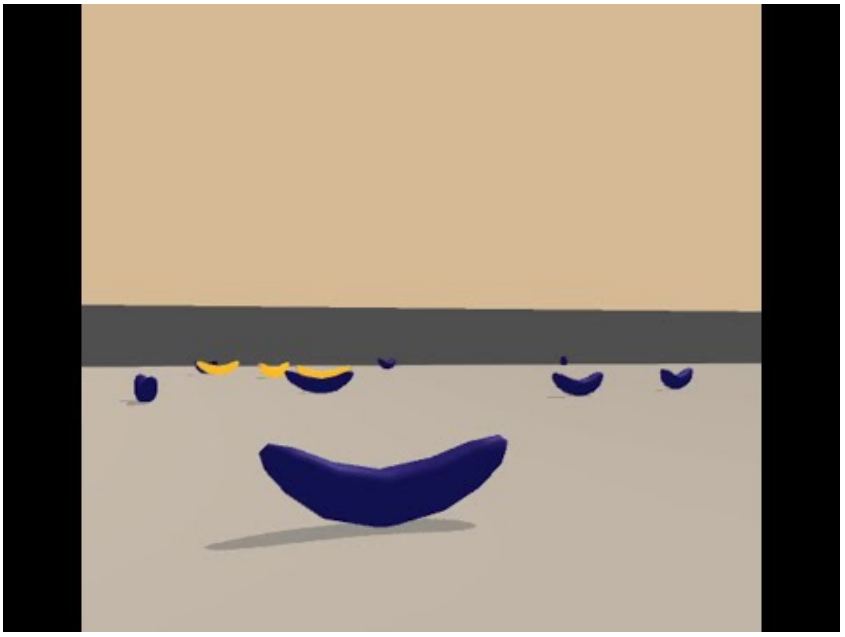
The trained agent achieves a high score also in the testing part as shown in the tested code and in the video.

The algorithm uses the prioritized an replay, it stores a set of tuples of experiences in a replay buffer which the agent can sample and lean by reusing experiences from the past.

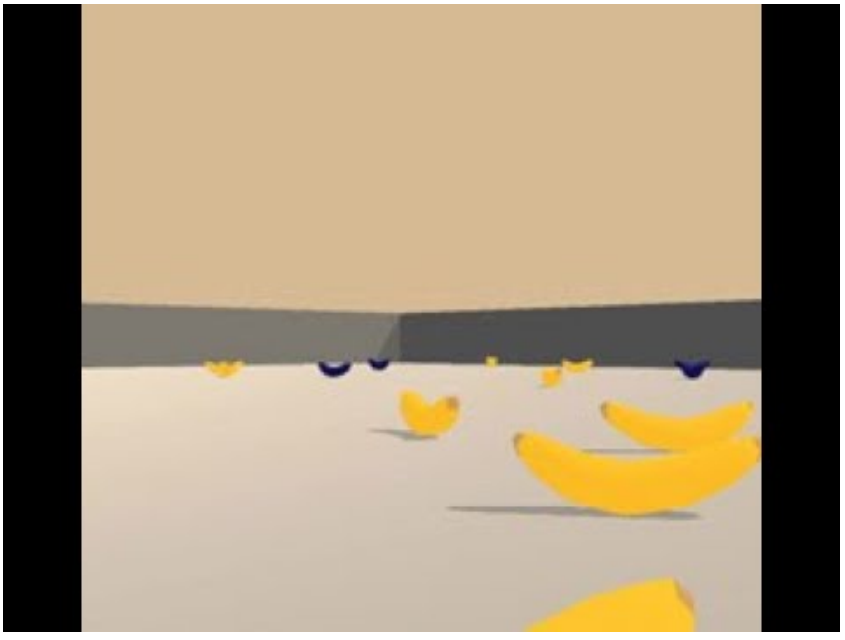
Video of the agent: watch the results

I recorded two videos of my agent, the untrained and trained agent.

Video of the untrained agent



Video of the trained agent



Algorithm improvements

The algorithm solves a simple game but if it were used for a more difficult task several improvements could be added:

- a Double DQN
- a Dueling DQN
- Prioritized experience Replay

Prioritized experience Replay

The prioritized experience replay assign significance values to each stored tuples to give a priority.

Double DQN

Q-learning is prone to overestimation of the action values because is based on the maximization steps which selects higher values without trying other possible values. Are these values trustful? The mechanism of double DQN provides a robust counterpoint, with an evaluation step which relies on another set of values. If the two mechanisms are not agreeing on the maximum, the value set is lower. These prevents propagating of false maximum values obtained by chance.

Dueling DQN

The dueling DQN uses two streams, one estimates the state value function, the other the advantage for each action. The two branches are in the fully-connected layers. The Q-value is a combination of the two values.

Project Starter Code

The project starter code of the original Udacity repo for this project can be found here.