Report

Learning Algorithm (using DQN with experience replay)

initialize replay buffer D size to N

initialize qnetwork_local with weights θ

initialize qnetwork_target with weights $\theta^{-}=\theta$

For episode =1, n_episodes do

Initialize state s_1

For t=1, max_t do

With probability ε select a random action a_t

Given s_t, a_t get $r_{t,}\, s_{t+1}$

Store transition (s_t,a_t, $r_{t,}\, s_{t+1}$) in D

Sample random minibatch of transitions (s_t,a_t, $r_{t,}\, s_{t+1}$) from D

Set $y_j=\begin{cases} r_j & if\ \ episode\ terminate\ at\ step\ j+1 \\ r_j+\gamma max_{a'}\hat{Q}(s_{t+1},a';\ \theta^{-}) & otherwise \end{cases}$

Perform a gradient descent step on $(y_j-\mathrm{Q}(s_t,a_j;\theta))^2$ with respect to the θ

Every C steps do soft update $\theta^{-}=\tau*\theta+(1-\tau)*\theta^{-}$

EndFor

Decrease ε

EndFor

Hyperparameters

batch_size = 64

eps = 1.0

eps_end = 0.01

decay = 0.999

max_memory_size = 100000

gamma = 0.99

alpha = 5e-4

tau = 1e-3

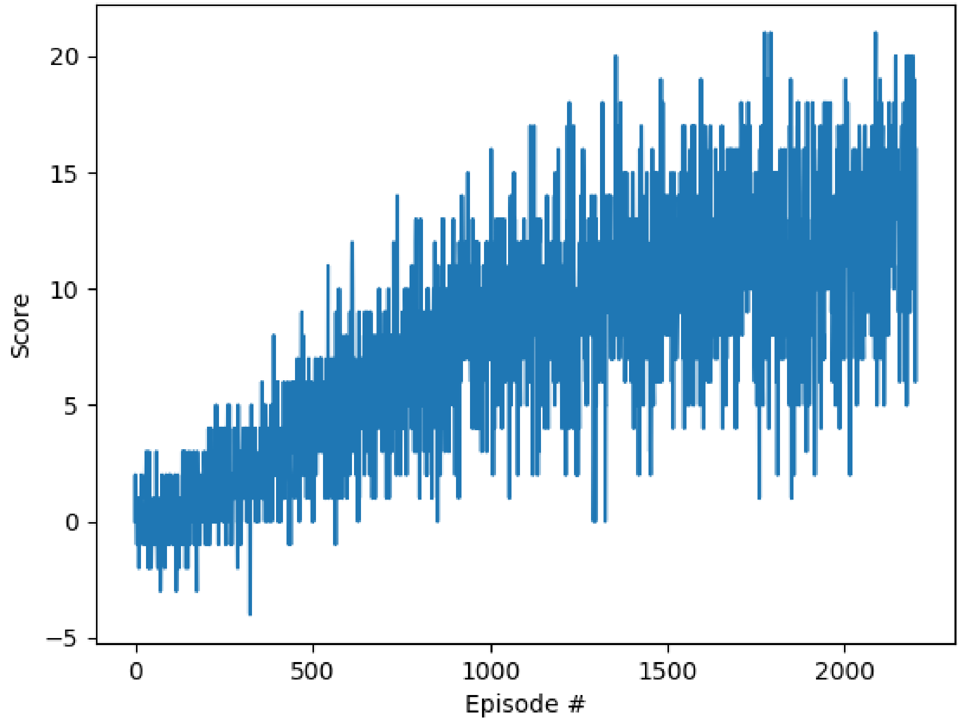
update_every=4

max_t=1000

Model Architecture

Input: state_size→fc1:128 →ReLU→fc2:64→ReLU→fc3:action_size

Plot of Rewards



Episode: 100,   Average Score: 0.0198

Episode: 200,   Average Score: 0.4200

Episode: 300, Average Score: 1.4400

Episode: 400, Average Score: 2.5000

Episode: 500, Average Score: 3.2200

Episode: 600, Average Score: 4.4300

Episode: 700, Average Score: 5.3400

Episode: 800, Average Score: 6.3300

Episode: 900, Average Score: 6.8200

Episode: 1000, Average Score: 8.2900

Episode: 1100, Average Score: 8.9200

Episode: 1200, Average Score: 9.3700

Episode: 1300, Average Score: 9.2800

Episode: 1400, Average Score: 10.3000

Episode: 1500, Average Score: 10.4700

Episode: 1600, Average Score: 11.0400

Episode: 1700, Average Score: 11.2100

Episode: 1800, Average Score: 11.8800

Episode: 1900, Average Score: 10.8300

Episode: 2000, Average Score: 11.4600

Episode: 2100, Average Score: 12.2300

Episode: 2200, Average Score: 13.1800

Future Work

In order to solve the DQN's problem about overestimating action values, using double Q-learning can be a better choice.

It can also use prioritized experienced replay buffer to learn more effectively instead of sampling experience transitions uniformly from a replay memory. The intuition behind that is the more important transitions should be sampled with higher probability.

By replacing the traditional Deep Q-Network (DQN) architecture with a dueling architecture, we can assess the value of each state, without having to learn the effect of each action. The value of most states don't vary a lot across actions, thus it makes sense to try and directly estimate them. Meanwhile it still need to capture the difference actions make in each state where the advantage function comes in.