Implementation of the DQN algorithms

The base DQN algorithm

For solving the banana problem I trained two agents. The first agent is based on a Deep Q-Network, and during training uses **experience replay** to sample from it randomly, and **fixed Q-targets** to avoid harmful correlation between the target and actual value of the Q-Network.

The Double DQN algorithm

As Deep Q-Learning tends to overestimate action values, I implemented **Double Q-Learning**, which has been shown to work well in practice to help with this. The Double DQN algorithm is a modification if the base DQN.

The training parameters

- maximum step during episodes: max_t=1000
- exploration rate start: eps_start=1.0
- minimum exploration rate: eps_end=0.01
- epsilon decay: eps_decay=0.995
- replay buffer size: BUFFER_SIZE = int(1e5)
- minibatch size: BATCH_SIZE = 64
- discount factor: GAMMA = 0.99
- for soft update of target parameters: TAU = 1e-3
- learning rate: LR = 5e-4
- how often to update the network: UPDATE_EVERY = 4

The Neural Network

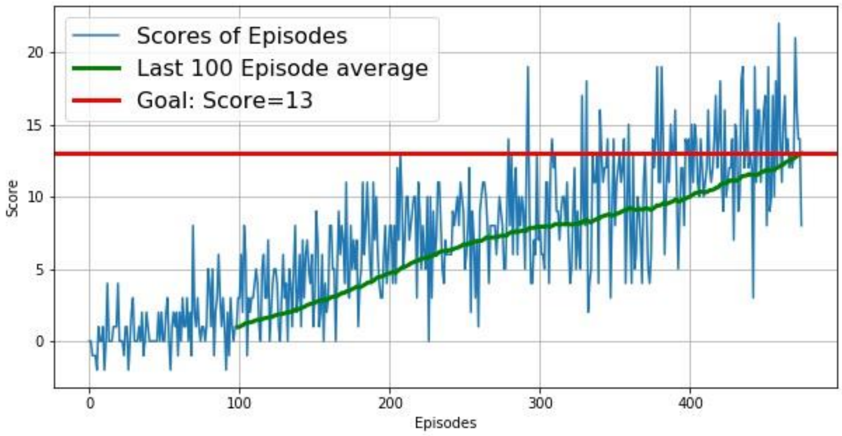
Behind the DQN algorithms there is a simple neural network to estimate the Q values.

This network has an input dimension of 37 as this is the state space dimension. The hidden layers size is 64. The size of the output layer is 4 as this is the number of possible actions.

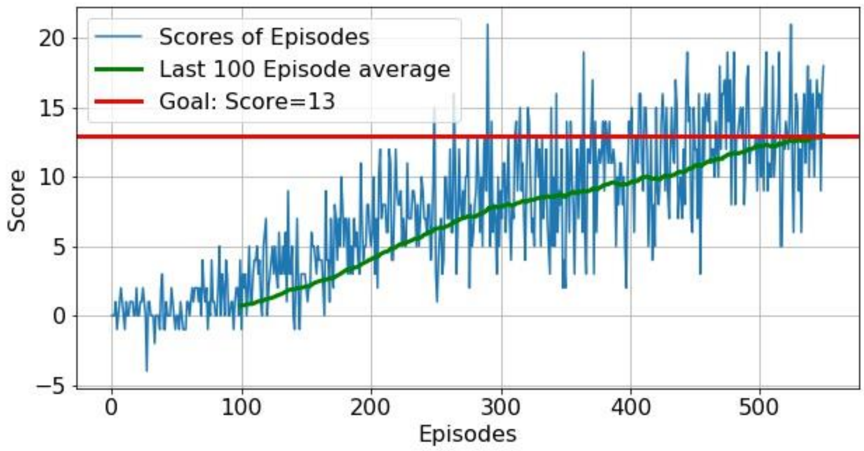
Results

The goal of the project was to train an agent to achieve an average score of +13 over 100 consecutive episodes. With my neural-network architecture and hyperparameters the non-double DQN achived this goal during less than 500 episodes, while the Double DQN during more than 500 episodes, but the difference wasn't large.

The non-Double DQN:



The Double DQN



Possible Improvements

- optimize **hyperparameters**
- implement **prioritized experience replay**
- implement **Dueling DQN**