Project 1: Navigation

Learning Algorithm.

The learning algorithm used will be DQN. This value-based method uses a NN-model which estimates Q(s,a) by inputing the states and outputting a value for each action.

Parameter(weight) fitting is done by minimising the MSE between the target Q(s,a) and the current Q(s,a). This is iteratively obtained, epic by epoch, through `loss.backward()` in PyTorch.

In DQN the Q(s,a) is calculated as the next reward and the estimation of the next highest Q action. This causes overestimation bias. One of the solutions for future work is implementing double q-learning.

We will use the code exercise done in the DQN exercise as baseline. Those hyperparams fulfilled the requirements. So I decided to save GPU time for the time being. Once I've succesfully done the other two tasks, and if I have GPU time available I'll try implementing the ideas I address in last section of report.

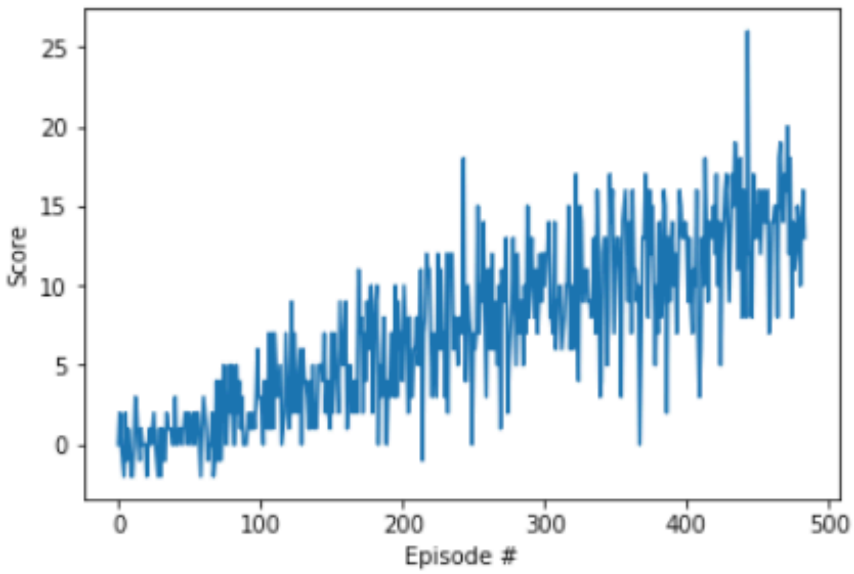
Therefore, hyper parameters defining the algorithm are:

- BUFFER_SIZE = int(1e5)
- BATCH_SIZE = 64
- GAMMA = 0.99
- TAU = 1e-3
- LR = 5e-4
- UPDATE_EVERY = 4
- n_episodes = 484
- max_t=1000
- eps_start=1.0
- eps_end=0.01
- eps_decay=0.995

The NN used, like in the dqn exercise has 2 64-neurons hidden layers. Input layer has state(37) inputs and output layer has the actions(4)

Plot of Rewards

```
Episode 100	Average Score: 0.90
Episode 200	Average Score: 4.42
Episode 300	Average Score: 7.90
Episode 400	Average Score: 10.62
Episode 484	Average Score: 13.02
Environment solved in 384 episodes!	Average Score: 13.02
```



Ideas for Future Work

Algorithm
Implementing what they call Rainbow in Deep Mind. In particular, I'm really interested about the way modelling the Q(s,a) as a distribution could help.

Model
Try both way simpler and way more complex architectures.