Learning algorithm

To solve this problem the approach was to take the deep-Q network (DNQ) implementation from Udacity that was given in the LunarLander-v2 exercise and tweak it.

Because modifying the learn function to do not just DNQ, but also double DNQ was relatively easy and from the lessons it seemed that this would perform better this was done straight away with below modification.

```
# Get the max actions from the target network (not the Q-values)
# Shape: (batch_size,)
max_actions_next = self.qnetwork_local(next_states).detach().max(1)[1]
# Use those max actions to get the corresponding Q-values from the target network
# Shape: (batch_size, 1)
Q_targets_next = self.qnetwork_target(next_states).gather(1, max_actions_next.unsqueeze(1))
```

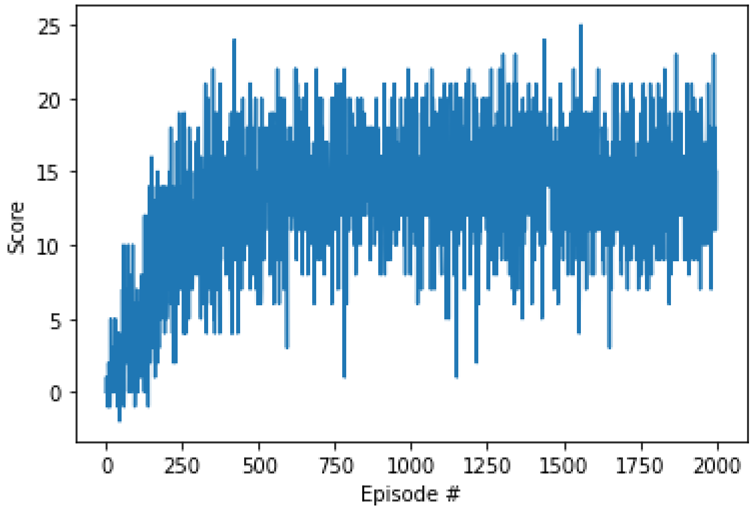
The parameters used for the Lunar lander however turned out not to solve the problem within 2000 steps, so various combinations were tried, most of them failed to give any improvement.

But eventually the solution was reached in with double DNQ, in Episode 700 using these parameters:

- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 128 # minibatch size
- GAMMA = 0.98 # discount factor
- TAU = 1e-3 # for soft update of target parameters
- LR = 9e-4 # learning rate
- UPDATE_EVERY = 8 # how often to update the network
- eps_start=1.0, eps_end=0.1, eps_decay=0.97
- fc1_units=128, fc2_units=128

After reducing the number of units in the network to fc1_units=64, fc2_units=64 the solution was reached even reached after 349 steps. So see if the algorithm would improve further after that it was kept running and the result saved each time it was better than the previous best. The highest score this way found was 14.96 at episode 1500.

Plot of rewards




```
Episode 100	Average Score: 2.29
Episode 200	Average Score: 6.67
Episode 300	Average Score: 10.70
Episode 400	Average Score: 12.12
Episode 449	Average Score: 13.05
Environment solved in 349 episodes!	Average Score: 13.05
Episode 500	Average Score: 13.28
Episode 600	Average Score: 13.89
Episode 700	Average Score: 14.52
Episode 800	Average Score: 14.27
Episode 900	Average Score: 14.75
Episode 1000	Average Score: 14.38
Episode 1100	Average Score: 14.48
Episode 1200	Average Score: 14.64
Episode 1300	Average Score: 14.34
Episode 1400	Average Score: 14.62
Episode 1500	Average Score: 14.96
Episode 1600	Average Score: 14.87
Episode 1700	Average Score: 14.31
Episode 1800	Average Score: 14.00
Episode 1900	Average Score: 14.62
Episode 2000	Average Score: 14.42
```

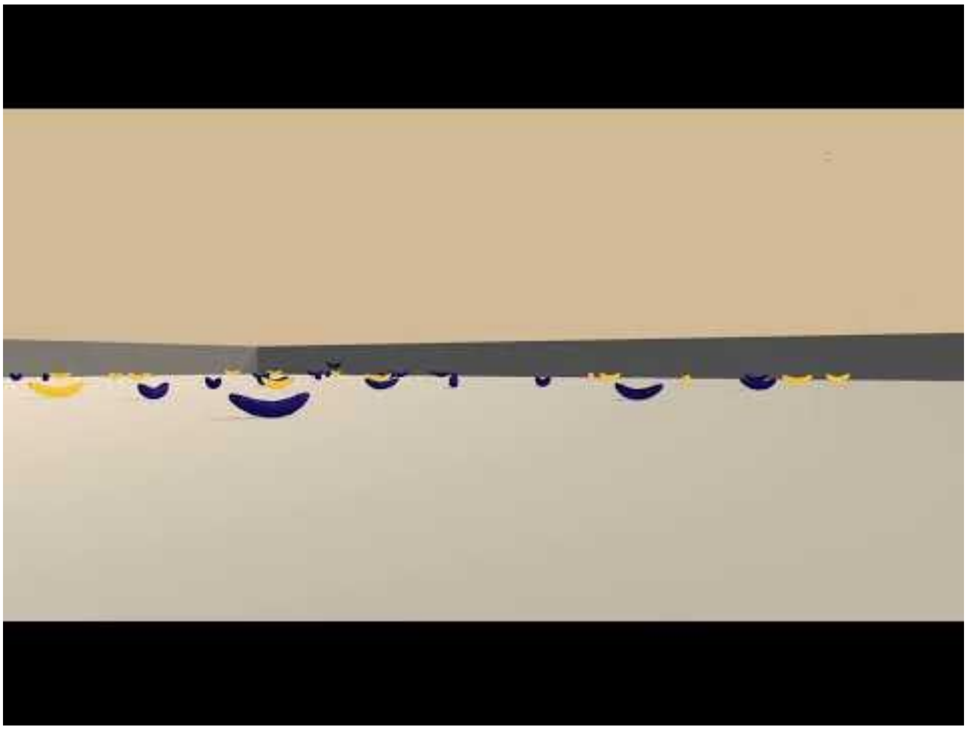
Ideas for future work.

1) The first idea would be to do an exhaustive grid search to find the optimal set of hyper parameters. There are so many combinations to chose from that it's unlikely that the ones found are the optimal ones. Unfortunately even testing a subset of these combinations is exhaustive in time consumption.

2) Investigate bad corner cases

The saved weights were loaded and run a couple of time in non-training mode.

Observing some of the episodes that gave extreme low results (one going as low as '2' showed that the algorithm sometimes seems to get stuck in going back and forth between 2 or 3 states. As can be seen in below video.



Perhaps it would work to keep track of the x past states when simulating an episode and ignore the action suggested by the trained network when a tight closed loop between states is detected and instead chose another random action to break out of it.

Alternatively a priority based replay buffer might be investigated to check if this solves these cases.

3) The dueling DQN algorithm suggested by udacity could be tried.