Learning Algorithm

The algorithm used was DDPG due to the fact that the action space in this case is continuous. The two agents in the environment were trained using a single shared actor and critic due to the symmetry of their observations. Note that additionally noise in the form of a normal distribution was added to each action value in order to aid with exploration.

Note that this code is adapted/inspired from CleanRL's implementation, with attributions in the code where necessary
https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/ddpg_continuous_action.py
This is in line with the implementation of Project 2

Hyperparameters

The hyperparameters used are listed below

```
args.buffer_size = int(10000)
args.gamma = 0.99
args.tau = 1e-2
args.max_grad_norm = 0.5
args.batch_size = 128
args.exploration_noise = 0.5
args.learning_starts = args.buffer_size
args.policy_frequency = 5
args.noise_clip = 0.5
args.total_episodes = 10000000
args.learning_rate = 1e-4
```

Note in particular that the learning only starts after a "warm up" period of 10,000 steps needed to fill the buffer.

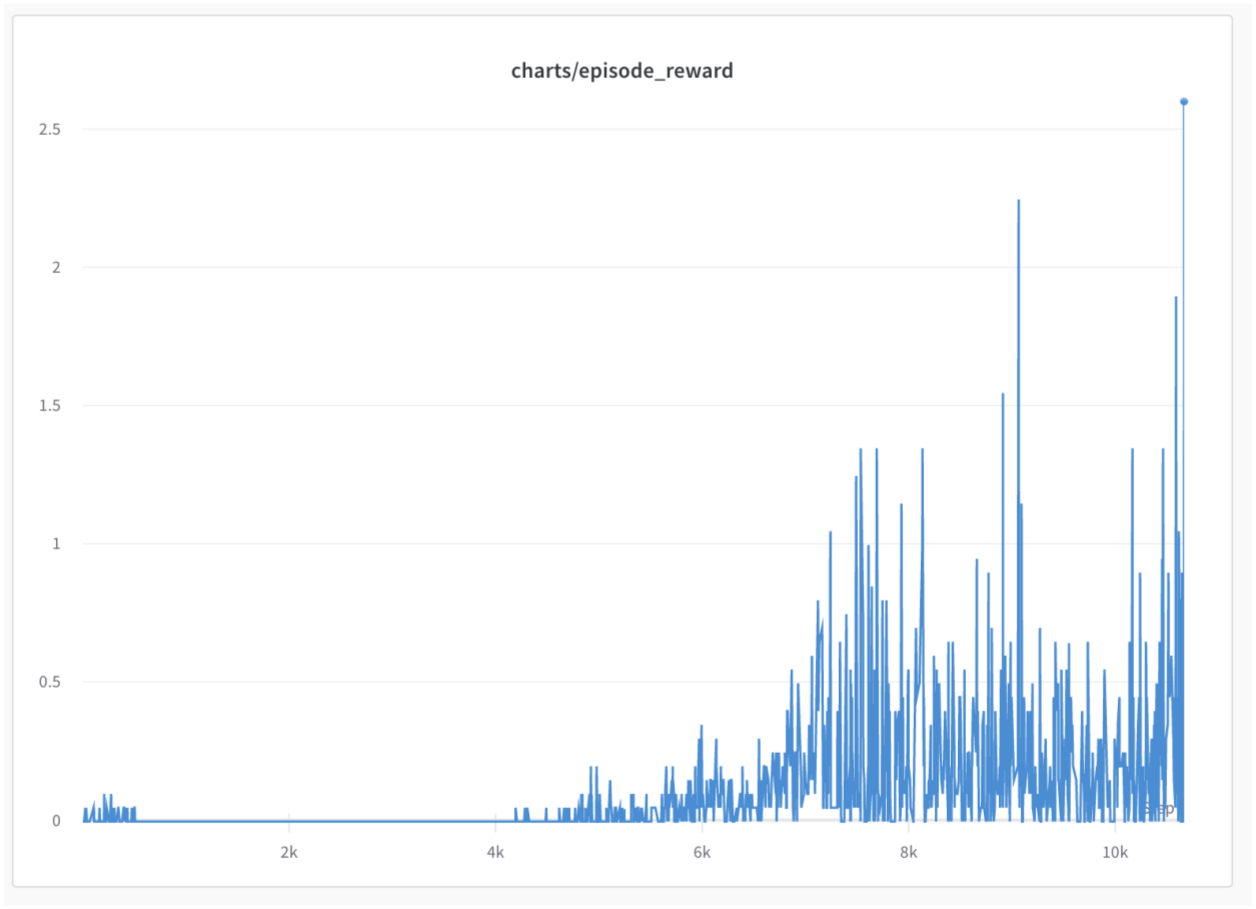
Model Architectures

There were two neural networks involved here:

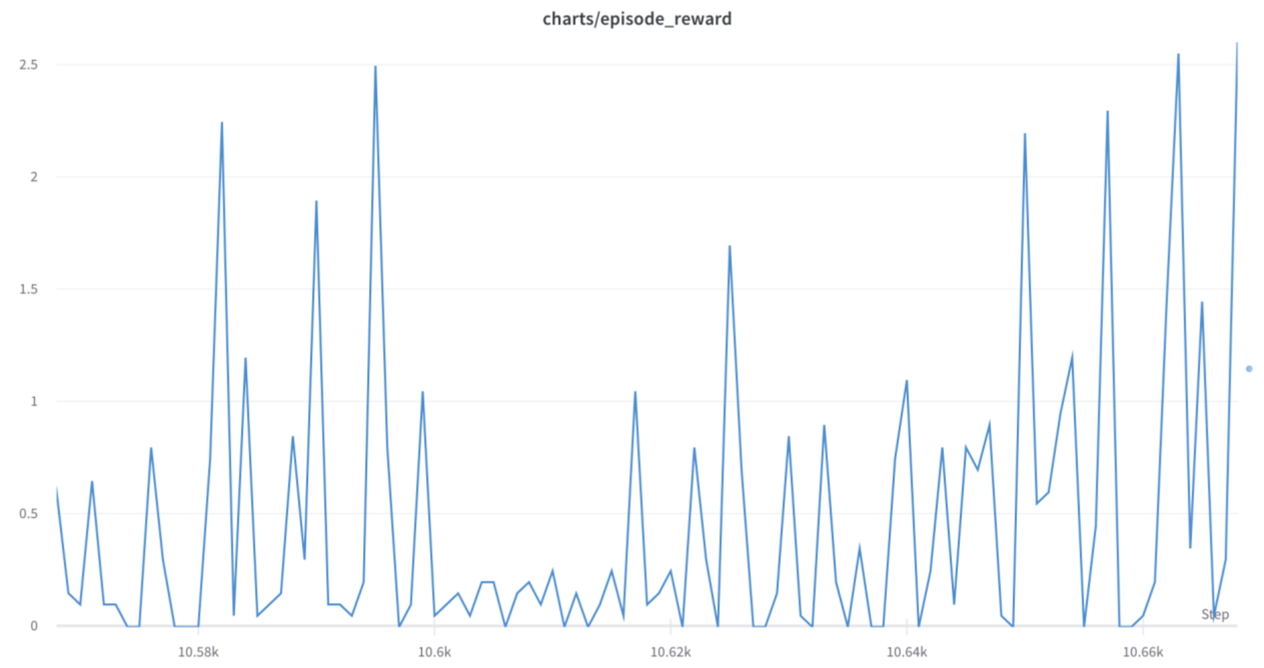
1. The Critic network. This was used to estimate the value of each taken action. This took as input into the first layer both the current action and the current state. This was followed by two consecutive hidden layers of 256 ReLU neurons each, followed by a linear output neuron.
2. The Actor network. This was used to produce the actions to be taken, given the state at each point. This had two consecutive layers of 256 ReLU neurons each, with an output layer producing one tanh-capped action value for each action in the space.

Note that the actor and critic networks were identically shared between both of the agents in the environment

Results



The above chart denotes the final reward per episode. As you can see, the environment is solved by episode 10,668, producing an average reward of 0.5 across the preceding 100 episodes. Below you can see a highlight of these final successful 100 episodes



Future Work

The efficiency of training could be increased by implementing Self Adaptive Double Bootstrapped DDPG, for example https://www.ijcai.org/Proceedings/2018/0444.pdf
This may increase the speed with which the environment is solved.

Additionally, since this environment contains multiple (albeit identical) agents, MADDPG could also be used to allow the agents to learn to work in a coordinated rather than independent manner