Description of implementation
DDPG implementation methodologies and hyperparameters were based on Shangtong's work, coursework and assignments.

DDPG uses 4 neural networks, an actor who is responsible for choosing the best actions and a critic who is uses TD to learn value functions. 4 networks exist since the actor and critic each have a neural net that learns and another network that periodically gets updates from the learning net. The idea is to reduce the odds of training on data that you are actively learning about. The Critic tries to minimize the difference between the expected values of the next actions taken by the actor and the directly calculated critic values.

Batch Normalization
I went through a dozen different implementations before realizing that batch normalization was not implemented correctly. This paper https://arxiv.org/pdf/1502.03167v3.pdf describes the importance. Batch normalization was setup before any fully connected layer where internal covariate shift could occur.

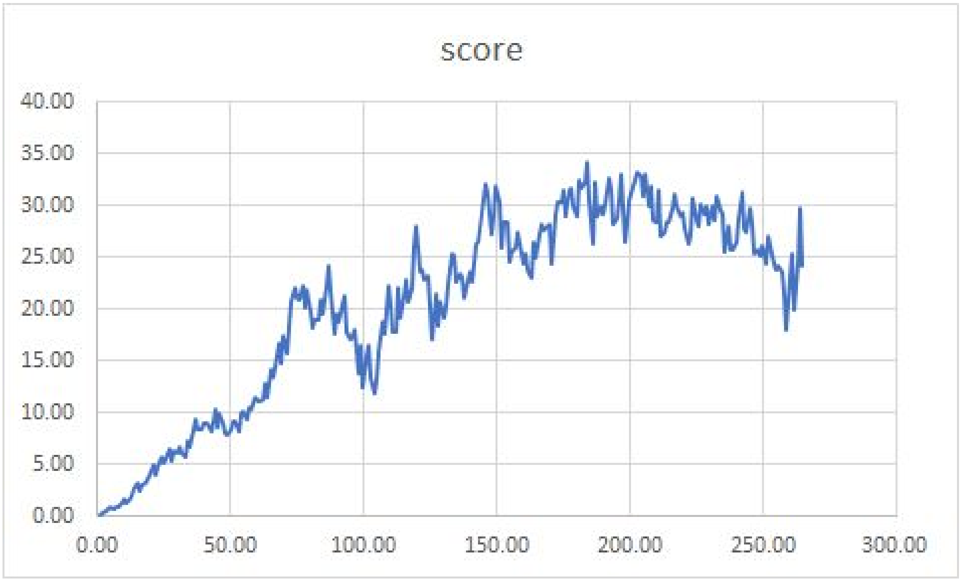
Replay Buffer
1e6 was used. I experimented with reducing it, theorizing it would result in faster learning since newer collected should have actions associated with positive outcomes. I saw overfitting (admittedly batch normalization was not correct) so switched it to 1e6. At worst training will be slower if this is large.

Network architectures
Both the actor and critic use 2 hidden layers with 128 nodes in each layer.

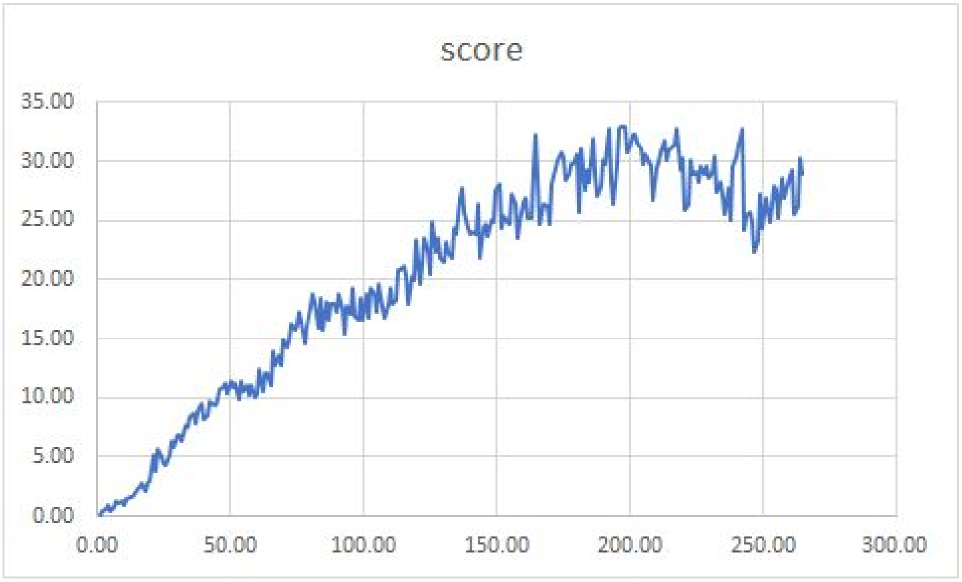
Learning rate
Both the actor and critic use 1e-3. Batch normalization allows this to be higher than it would have to be otherwise.

Update frequency
20 updates every 20 steps. Reducing the learn updates and frequency of learning is expected to reduce the score declines thereby reducing the episodes to solve.

1st attempt

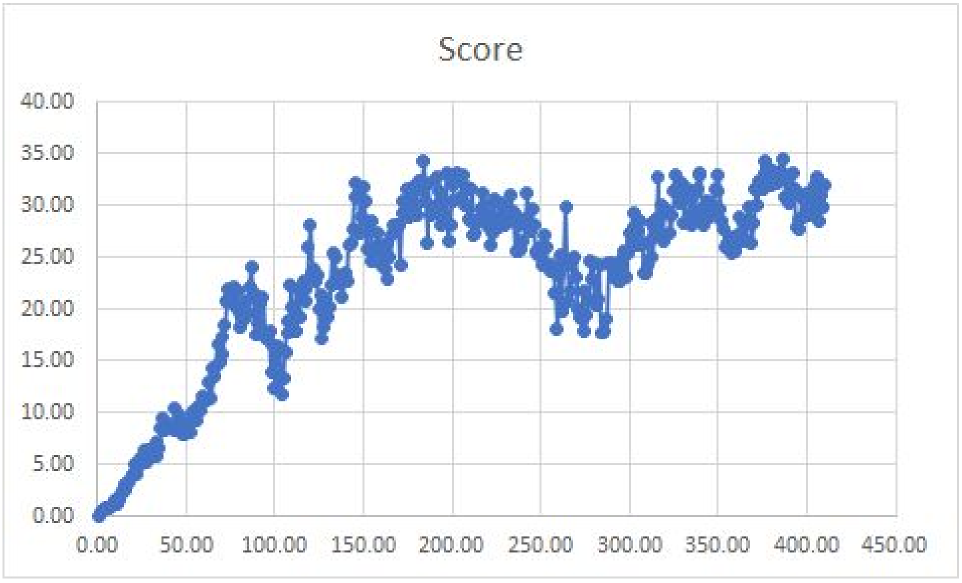


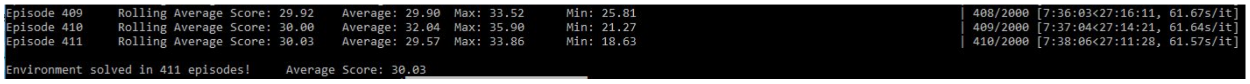
Changes made: Removed the actors 2nd hidden layer and reduced the critics 2nd hidden layer to 64 and only do 10 learn steps on each update.
Result: There are fewer drops but at 200 it took a swing down.



Changes: Reduce the actor's learning rate to 1e-4
Result: It got stuck at ~9
Changes: Reduce the frequency of learning to once every 40 steps and use an architecture of 128 and 64 for the actor and critic.

I checked back on the 1st model and after 7hrs and 38 minutes and 411 episodes it eventually solved the environment.





```
Episode 409     Rolling Average Score: 29.92    Average: 29.90  Max: 33.52      Min: 25.81                                      | 408/2000 [7:36:03<27:16:11, 61.67s/it]
Episode 410     Rolling Average Score: 30.00    Average: 32.04  Max: 35.90      Min: 21.27                                      | 409/2000 [7:37:04<27:14:21, 61.64s/it]
Episode 411     Rolling Average Score: 30.03    Average: 29.57  Max: 33.86      Min: 18.63                                      | 410/2000 [7:38:06<27:11:28, 61.57s/it]

Environment solved in 411 episodes!     Average Score: 30.03
```

Ideas for Future Work:

- Take Shangtong's work and integrate tensorboard so that it is easier to tell what is happening with model training.
- Create a visualization that shows what is driving the 50 epic score declines.
- Use AWS or Google to tune hyperparameters.
- Find a robot that could be simulated and trained and then transfer-learn it to the robot.