Continuous Control:

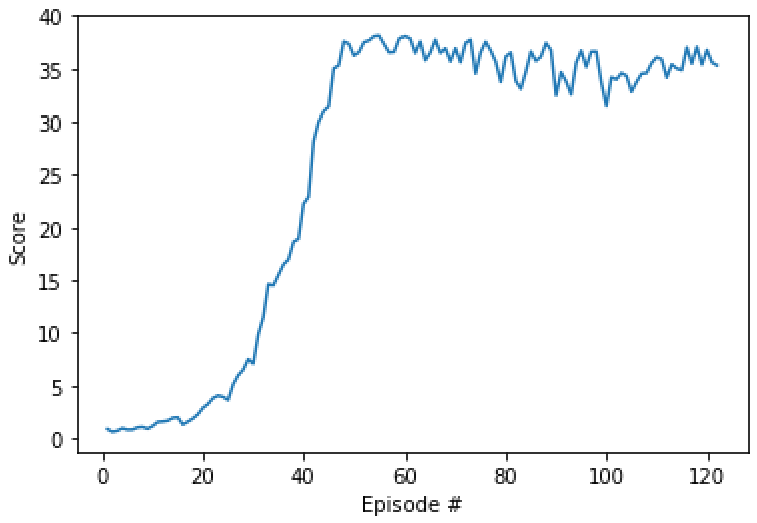
This document describes how the deep reinforcement learning Nano-degree project 2 from Udacity is implemented. For this project, version two (**20 Agents**) is implemented.

Deep Deterministic Policy Gradient algorithm is used for this project implementation where there are two deep network, Actor and Critic. Both have two hidden layers with 256 nodes in first hidden layer and 128 nodes in the second hidden layer. The is similar to what is being used in the DDGP architecture that were used in the Udacity class example.

The following are the parameters that I used.

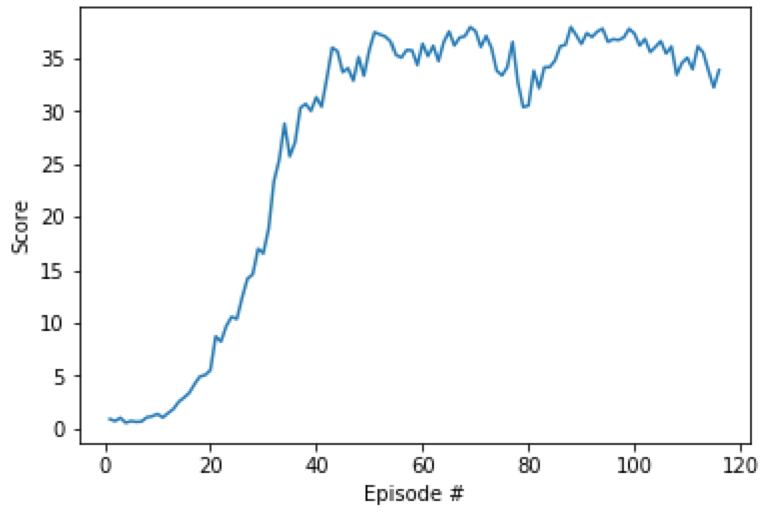
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-4        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```

With my current implementation, 122 episodes is needed to converged. This is achieved by trying many different combination of hyper-parameter along with many attempt of different number of nodes in both actor/critic hidden layers. This also require a lot of code debugging and reading Pytorch documentation. Also, this was a very time consuming process. However, the result is a nice smooth curve and the network has simply learned and converged at about 46 episodes. This is reasonably fast in my view. And when I test my result, I got an average of above 36 for my score.



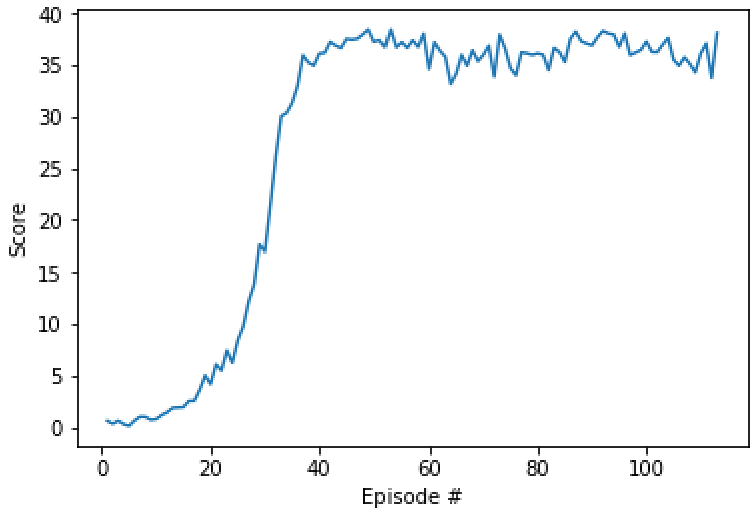
Possible improvement:

One of the possible improvement is to use Batch normalization. This should be able to speed up the converge time. However, when I tried it in my jupyter notebook with batch normalization, I did not see a lot of improvement. Here is the plot.



It may have converged 1-2 episodes earlier compare to my previous plot. However, after it converged, the fluctuation is slightly larger. And in my view, I am not sure this improvement is worth the effort.

I also tried to change the random seek value from 4 to 10. And I did see little more improvement than adding normalization. Here is the Plot that I collected.



It coverages before episode 40 and have about the same fluctuation than my original code. I have had experience that different seek value can change the result a bit. I am not sure this is due to Pytorch by Facebook is not matured or there are other issues that I did not aware of.

Other than fine-tuning all the possible parameters. Algorithms such as Advantage Actor Critic (A2C) and Asynchronous Advantage Actor Critic (A3C) can be used to improvement overall performance.