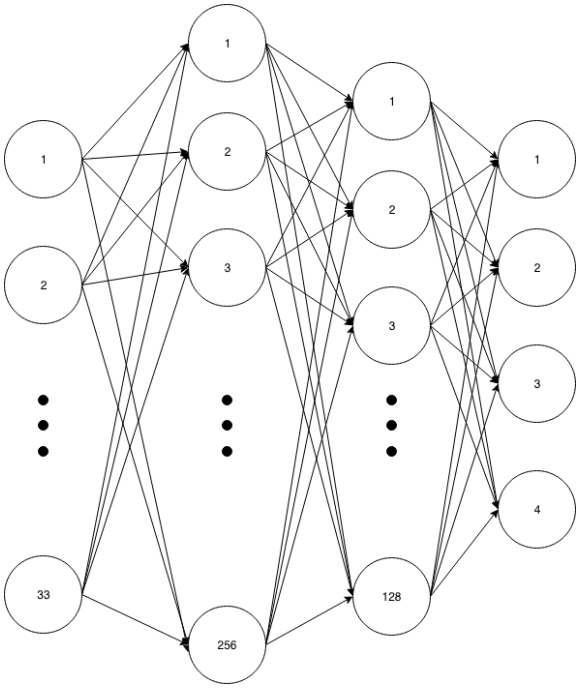
Mathias Babin
10/9/2018

P2 – Continuous Control Report

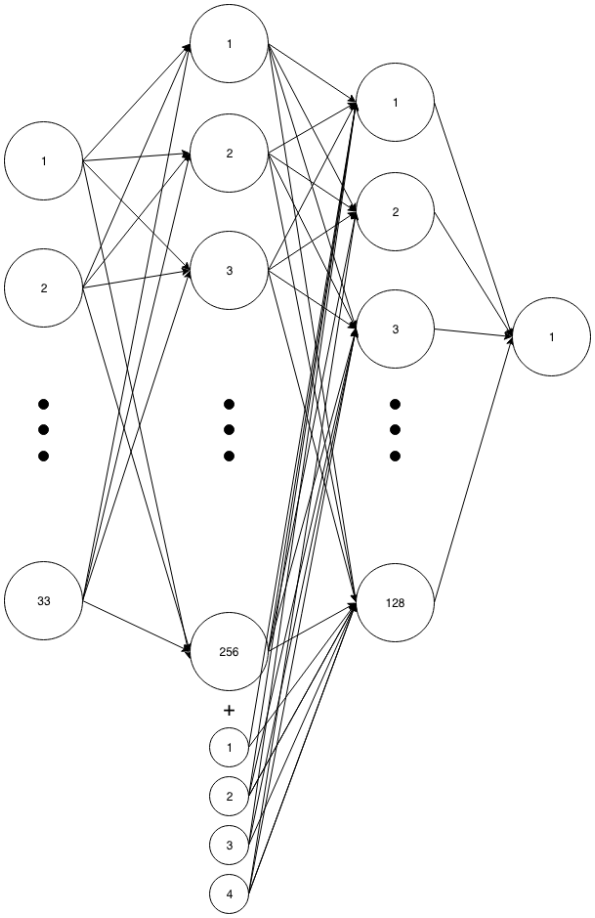
1) Learning Algorithm

For this project I implemented the DDPG algorithm, with improvements made via Priority Experience Replay (PER), and Parameter Noise. The Neural Network(NN) architecture for this project includes 33 input neurons to reflect the environment's state space, 256 and 128 hidden units, and 4 outputs for the actor networks and 1 output for the critic networks. It should be noted that the second hidden layer for the critic networks include 4 additional units to account for the four outputs produced by the Actor networks. The learning rate for all NNs was 0.0001.

Actor Networks:



Critic Networks:



PER was implemented using a sum tree data structure for the replay buffer, as it achieves O(log n) time for sampling. PER's coefficient 'a' was set to 0.95 as I wanted sampling to mostly occur on experiences with high priority values, and epsilon was set to only 0.01. The size of this replay buffer was 100 000, and batches contain 80 experiences. Finally, as suggested in an OpenAI blog post(https://blog.openai.com/better-exploration-with-parameter-noise/), I added a parameter noise to all four NNs via a parameter noise layer.

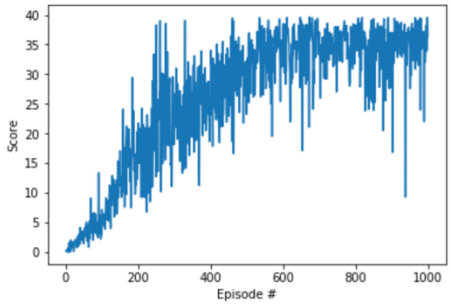
It should also be noted that a soft update rules was used to update the weights of the NNs where Tau was set to 0.01, where updates occurred every timestep.

2) Results

Included with this report are two sets of results: the first without PER; and the second with PER. Both sets already contained parameter noise, and no other hyperparameters changed.

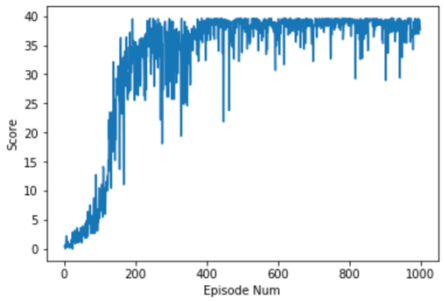
Random Sampling:

```
Episode 100     Average Score: 2.90     Score: 7.17
Episode 200     Average Score: 11.84    Score: 21.75
Episode 300     Average Score: 19.88    Score: 18.15
Episode 400     Average Score: 24.14    Score: 23.15
Episode 500     Average Score: 28.95    Score: 23.24
Episode 600     Average Score: 33.21    Score: 38.19
Episode 700     Average Score: 34.68    Score: 33.98
Episode 800     Average Score: 34.64    Score: 35.97
Episode 900     Average Score: 34.15    Score: 37.01
Episode 1000    Average Score: 35.13    Score: 36.33
```



PER:

```
Episode 100     Average Score: 2.96     Score: 6.03
Episode 200     Average Score: 13.11    Score: 35.57
Episode 300     Average Score: 20.10    Score: 38.79
Episode 400     Average Score: 30.82    Score: 39.54
Episode 500     Average Score: 35.71    Score: 33.71
Episode 600     Average Score: 37.06    Score: 35.91
Episode 700     Average Score: 38.11    Score: 37.30
Episode 800     Average Score: 38.40    Score: 39.57
Episode 900     Average Score: 38.48    Score: 37.85
Episode 1000    Average Score: 38.40    Score: 37.68
```



As can be seen from these two graphs, not only did PER help the agent train much faster, but its behaviour was much more consistent as its average score has far less variance (especially near the end). Also during both of these training runs, the average score was calculated using 300 consecutive episodes as opposed to 100, and as a result earlier episodes drastically reduced the overall average; This issue has been resolved, and the averages now properly reflect how well the agent is training by only looking at the last 100 episodes.

Using the PER training example, the episode seems to have been solved around the 200-300 episode mark as oppose the none-PER run where the episode was solved around the 500-600 mark.

3) Future Improvements

Although it is not necessarily an improvement for this particular environment but implementing the version of the project which requires 20 agents for training would likely reduce the number of episodes that are needed to train based on the benchmarks provided by Udacity. Also, I did not experiment much with the NN architecture for this project because I was training on an older Macbook Pro's CPU so I could not afford to test larger networks. Finally, my implementation of Parameter noise only effects one layer, perhaps adding more layers of noise could produce better results and I did not implement 'layer normalization' as suggested by Open AI in order to combat any layer sensitivities to perturbations, or the adaptive scheme they describe for adjusting the size of parameter space perturbations which certainly warrants further investigation.