Learning Algorithm

The DDPG algorithm was used on this challenge, modeling a similar approach used in the code sample for the BipedalWalker-v2 example. Further, when I got (really) stuck with an agent that didn't seem to be learning, I reviewed some writeups of DDPG implementations that solve other continuous control tasks, which were helpful in suggesting tactics like gradient clipping in the local critic network.

I only marginally changed the hyperparameters beyond the defaults, but there were many different variables beyond the hyperparameters (such as the use of noise added to the action space) that I found affected the ability for the agent to learn effectively and within a reasonable period of time. Generally, I do not perceive that the hyperparameters were as important as the architecture of the neural networks or the use of these other variables such as noise (but that will be covered in more detail below).

The Actor neural network architecture I used had three hidden layers, with the input and output mapping of: *<state_size(33)> → 512 → 256 → 64 → 16 (not parameterized) → <action_size(4)>*

I experimented with a few different architectures. I first tried two hidden layers (256x256, 512x256, 256x128) but then changed to a deeper network (unfortunately, in parallel with other changes, making it harder later to infer the impact) in order to investigate why the network wasn't training. After standardizing on the three hidden layers, the three hidden size parameters (512, 256, and 64) were changed on various runs in order to compare performance, with the last output layer mapping of 16 → 4 remaining consistent (and not exposed through the API in `model.py`) in the latter stages of testing.

Even though I did over 20 unique runs (training for option 2, with 20 agents), I achieved extremely good results in only four runs, achieving the objective values in between 20 and 30 total episodes (three of the four of those have been included in the documented results, as I failed to display the `scores` and `scores_window` results before accidentally overwriting the variable values on one of the runs!). I never expected to do so many unique runs, and so my documentation of them is inconsistent, as I only started logging the details (time, hyperparameters used, add_noise on or off, seed value, etc) in the latter stages. However, in the next section I have included what I have in my notes.

The Critic neural network architecture I used also had three hidden layers. I never seriously investigated having a separate architecture for the Critic network, even though some of the documented results I have observed employ different sizes (in number of hidden layers and neurons per layer). Instead, I used the same basic architecture (same number of hidden layers, with parameterized number of neurons for each) for both Actor and Critic networks, even though the first hidden layer and the output layer necessarily change, given the focus of the Critic network.

As such, the Critic neural network architecture I used had input and output mapping of:
<state_size(33)> → 512 (`cat` to 516) → 256 → 64 → 16 (not parameterized) → 1*>.*

The hyperparameters used in the implementation were:

```
BUFFER_SIZE = int(1e6)  # 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-3        # learning rate of the critic
WEIGHT_DECAY = .0001    # L2 weight decay
```

The use of the WEIGHT_DECAY parameter is another area where – like the use of noise – I would like to spend more time in future analysis. The last runs I performed did NOT use it, but I expect it may have allowed results to be more stable.

Results and Plot of Rewards

Per above, training the agent was far more inconsistent (even after stabilizing the code) than I expected. For example, in the latter stages of testing, where I did almost 10 runs in a row with *exactly* the same code and configuration (and even the same initial seed), sometimes the agent would train well, and sometimes it wouldn't (and I would kill the job early to avoid the use of gpu time!). In total, I used about 15 hours of gpu time on all these runs, as the successful runs took over one hour (for 120-130 total episodes), and many of the unsuccessful ones were aborted early.

Further, I ended up getting these (inconsistent, but occasionally strong) results with `add_noise = False`, so at the end of the course (with any remaining gpu time!) I would like to experiment further with `add_noise = True` to see if I can achieve better.

Here is a (long) summary of the runs I did, with the output of the progress every 10 episodes (only included for the ones that were promising):

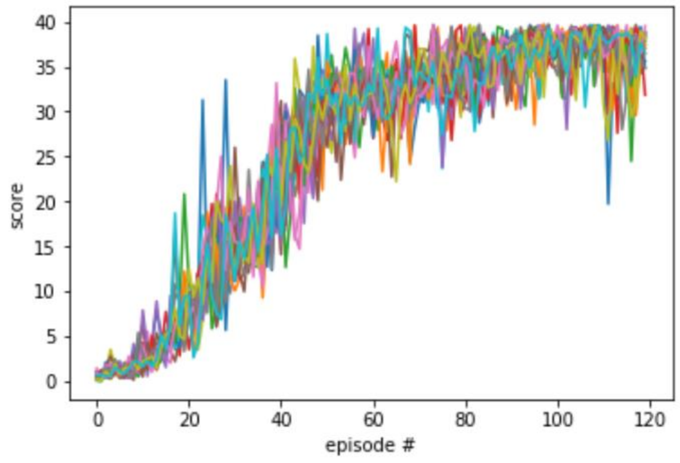
```
20 AGENT run1 1/1/2020 - validated it worked - killed
20 AGENT run2 1/2/2020 - great results after 36 min (23.118 after episode 80;
I should have let this one finish!!!!)
20 AGENT run3 1/2/2020 8.00am - killed (low results)
20 AGENT run4 1/2/2020 8.25am - killed (low results)
20 AGENT run5 1/2/2020 8.46am - killed (low results)
20 AGENT run6 1/2/2020 9.10am  model.py (converted), ddpg_agent (converted
from Agent2), changed add_noise = False
20 AGENT run7 1/2/2020 9.10am - killed (low results up to 140)

20 AGENT run8: 1/2/2020 start 10.25am
episode [10/500]        average score: 1.26
episode [20/500]        average score: 2.92
episode [30/500]        average score: 7.01
episode [40/500]        average score: 11.37
episode [50/500]        average score: 14.79
episode [60/500]        average score: 17.36
episode [70/500]        average score: 19.51
episode [80/500]        average score: 21.08
episode [90/500]        average score: 22.48
episode [100/500]        average score: 23.54
episode [110/500]        average score: 26.57
episode [120/500]        average score: 29.53
environment solved in 22 episodes... average score: 30.20
```

```
   ➔ files saved as km_actor20a.pth and km_critic20a.pth

20 AGENT run9: 4.17pm PST seed=0 - killed (scores too low)
20 AGENT run10: 4.26pm PST seed=12345 - killed (scores too low)
20 AGENT run11: 5.04pm PST seed=77777 - killed (scores too low)

20 AGENT run12: 5.10pm PST seed=0 - SOLVED!!!
episode [10/500]        average score: 1.13
episode [20/500]        average score: 2.74
episode [30/500]        average score: 5.23
episode [40/500]        average score: 8.52
episode [50/500]        average score: 11.21
episode [60/500]        average score: 14.11
episode [70/500]        average score: 16.44
episode [80/500]        average score: 18.14
episode [90/500]        average score: 19.80
episode [100/500]       average score: 21.14
episode [110/500]       average score: 24.53
episode [120/500]       average score: 27.59
episode [130/500]       average score: 30.18
environment solved in 29 episodes... average score: 30.18
   ➔ files saved as km_actor20b.pth and km_critic20b.pth

20 AGENT run13: 6.22pm PST seed=0 - 6.34pm - killed
20 AGENT run14: 6.45pm PST seed=0 - logic bug - killed
20 AGENT run15: 6.57pm PST seed=0 - killed (low scores)
20 AGENT run16: 7.04pm PST seed=0 - killed (low scores)

run17: 7.14pm PST seed=0 - finished 8.07pm
episode [10/200]        average score: 1.20
episode [20/200]        average score: 2.95
episode [30/200]        average score: 5.92
episode [40/200]        average score: 8.88
episode [50/200]        average score: 12.42
episode [60/200]        average score: 15.65
episode [70/200]        average score: 18.03
episode [80/200]        average score: 20.05
episode [90/200]        average score: 21.75
episode [100/200]       average score: 23.24
episode [110/200]       average score: 26.87
episode [120/200]       average score: 30.05
environment solved in 19 episodes... average score: 30.05
   ➔ files saved as km_actor20c.pth and km_critic20c.pth
```

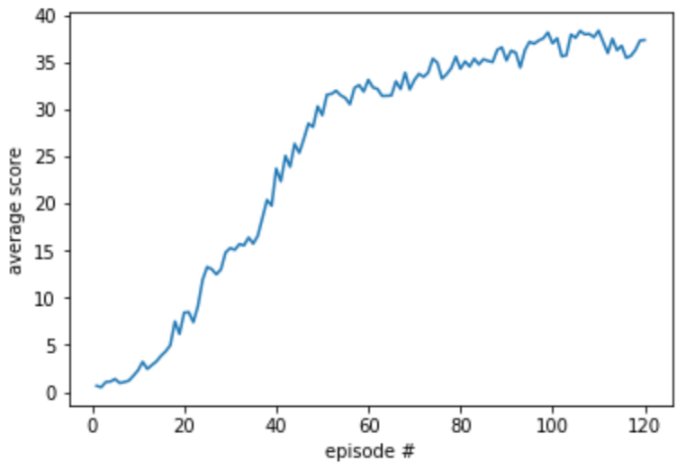
Here are more detailed results for run17, the last run that was catalogued here. After achieving these results I ran the validation cell to show that – using the trained agent – we received results as expected:

```
validation run:
total score (averaged over agents) this episode: 38.75699913371354
total score (averaged over agents) this episode: 37.71049915710464
total score (averaged over agents) this episode: 37.9209991523996
total score (averaged over agents) this episode: 36.546499183122066
total score (averaged over agents) this episode: 38.26749914465472
```

Here is the graph of the results of all 20 agents, output in the notebook:



Here is the graph of the average result over time for the entire set of agents, output in the notebook:



Lastly, I started a notebook later and was able to successfully load the `.pth` files from both Actor and Critic networks (even though I suppose I only need the Actor network) in order to show that the agents could be used to show a highly proficient trained agent:

```
actor file:  km_actor20a.pth
critic file: km_critic20a.pth
total score (averaged over agents) this episode: 32.990499262604864
total score (averaged over agents) this episode: 32.35449927682057
total score (averaged over agents) this episode: 32.11249928222969
total score (averaged over agents) this episode: 32.84799926578999
total score (averaged over agents) this episode: 30.93399930857122

actor file:  km_actor20b.pth
critic file: km_critic20b.pth
total score (averaged over agents) this episode: 34.08599923811853
total score (averaged over agents) this episode: 33.85099924337119
total score (averaged over agents) this episode: 33.65749924769625
total score (averaged over agents) this episode: 33.61099924873561
total score (averaged over agents) this episode: 34.79799922220409

actor file:  km_actor20c.pth
critic file: km_critic20c.pth
total score (averaged over agents) this episode: 37.977499151136726
total score (averaged over agents) this episode: 37.505999161675575
total score (averaged over agents) this episode: 38.368999142386016
total score (averaged over agents) this episode: 38.46249914029613
total score (averaged over agents) this episode: 37.456999162770806
```

Ideas for Future Work

Per above, I was surprised and dismayed at how unpredictable the training process was, even when using all the same code, hyperparameters and even seed values. There was an astounding amount of variation to the results that I can only attribute to the distribution in the initial locations of the spheres, and the randomness with which early (completely arbitrary) movements happened to correlate with sphere locations.

As such, I would be inclined to first investigate how randomness introduced by the `add_noise` parameter either helps or hurts the results. When I first started training (on the single agent version), since the agent wasn't learning effectively (inevitably for other reasons), the `add_noise` flag didn't seem to have a material effect. However, after making major changes (such as neural network architectures) and switching to the 20 agent challenge (in order to see if I was observing the same learning problems in that environment), I never went back to considering how `add_noise` would affect the results by turning it off or on: I was able to get decent results within a few runs, so I never tried it again. I would also like to more thoroughly try different learning rate and weight decay values during training, as I wrestled so long with some of the core code that I didn't pursue different permutations extensively.

After building a more robust (and more importantly, predictable) process for solving the problem, I would like to consider the strengths and weaknesses of other algorithms such as PPO, A3C or D4PG in solving the main problem. (I want to read the recommended paper that benchmarks different algorithms for continuous control tasks to consider this before attempting one of them.)