PROJECT SPECIFICATION

Continuous Control

Training Code

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Training Code	The repository includes functional, well-documented, and organized code for training the agent.	The training code is defined in the function ddpgrunner in cell <25> and the model is in model.py and the agent is in ddpg_agent.py
Framework	The code is written in PyTorch and Python 3.	The code is written in pytorch and python 3
Saved Model Weights	The submission includes the saved model weights of the successful agent.	The model weights are saved to checkpoint_actor.pth and checkpoint_critic.pth in local directory

README

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
`README.md`	The GitHub submission includes a `README.md` file in the root of the repository.	The git repo has the README.md file in the main directory.
Project Details	The README describes the the project environment details (i.e., the state and action spaces, and when the environment is considered solved).	These details are updated under “Project Details” section of the README.md file
Getting Started	The README has instructions for installing dependencies or downloading needed files.	These details are updated under “Getting Started” section of the README.md file
Instructions	The README describes how to run the code in the repository, to train the agent.	These details are updated under “Instructions” section of the README.md file

Report

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Report	The submission includes a file in the root of the GitHub repository (one of `Report.md`, `Report.ipynb`, or `Report.pdf`) that provides a description of the implementation.	This document is the report.pdf addressing all the rubric points. It discusses clearly all the events in the implementation of the project.
Learning Algorithm	The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.	The learning algorithm used is from the paper https://arxiv.org/pdf/1509.02971.pdf It is mentioned below this table.
Plot of Rewards	A plot of rewards per episode is included to illustrate that either: • ***[version 2]*** the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30. The submission reports the number of episodes needed to solve the environment.	Please take a look at the plots in cells 16, 26 and 27, 29, 30, 32 in the notebook.
Ideas for Future Work	The submission has concrete future ideas for improving the agent's performance.	The current work can be extended in several ways: 1. Increase the depth of the networks

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
		2. Vary the architecture and the optimizers of the networks. 3. Do more research on the hyper parameters. 4. Implement other algorithms such as PPO, A3C, and D4PG 5. Extend to other agents like crawler, single agent etc.

Learning Algorithm:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

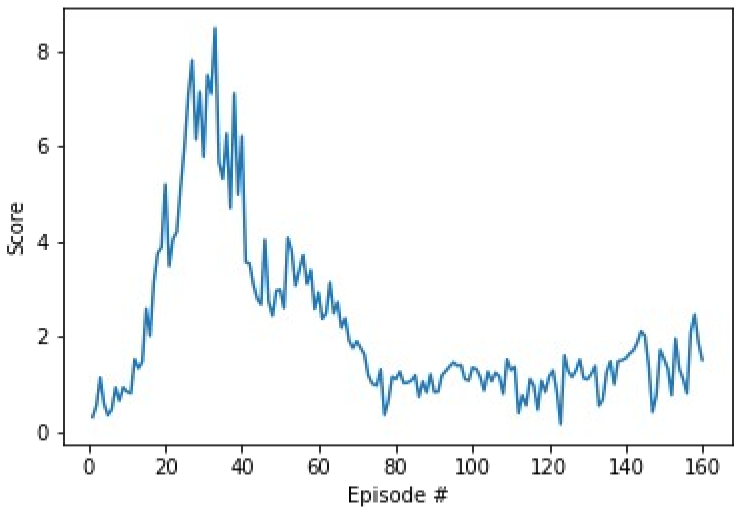
Implementation Details:

As hinted in the lessons for the projects, I started off with the DDPG pendulum implementation from the DRLND github repository. As per the hyper parameters the to start with I tried to stick with the parameters mentioned in the DDPG paper in section 7 (Experiment Details). Initially, I didnt have any luck with the results. I also stuck with the base model implemented in the ddpg pendulum implementation. I started seeing better results once I added batch norm layer to the actor model. After that I added, the gradient clipping as mentioned in attempt 3 of the benchmark implementation during lessons. I added it to both actor and critic. At this point of time i'm seeing the score go up but not closer to the target score of 30 even thought I trained for really many episodes as many as 500.

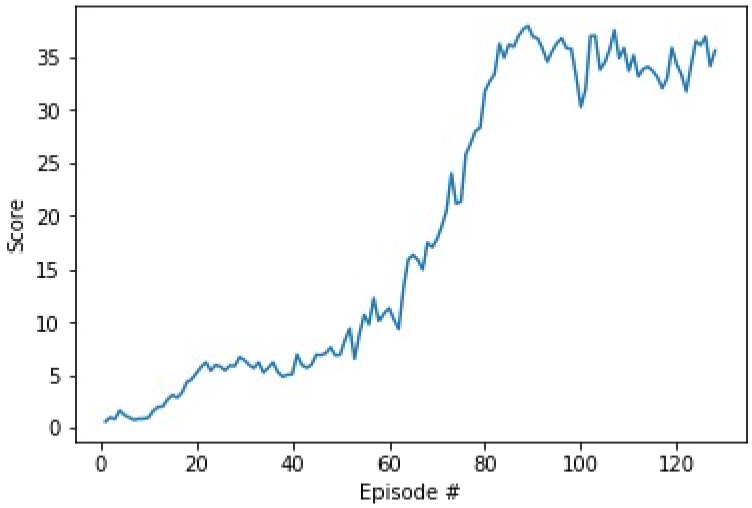
At this point of time I started playing with other parameters such as batch size, buffer size, learning rate. After playing around a bit with learning rate, I decided to keep the learning rate as per the paper for both actor and critic networks

The following are the results of initial tests with batch sizes and buffer sizes.

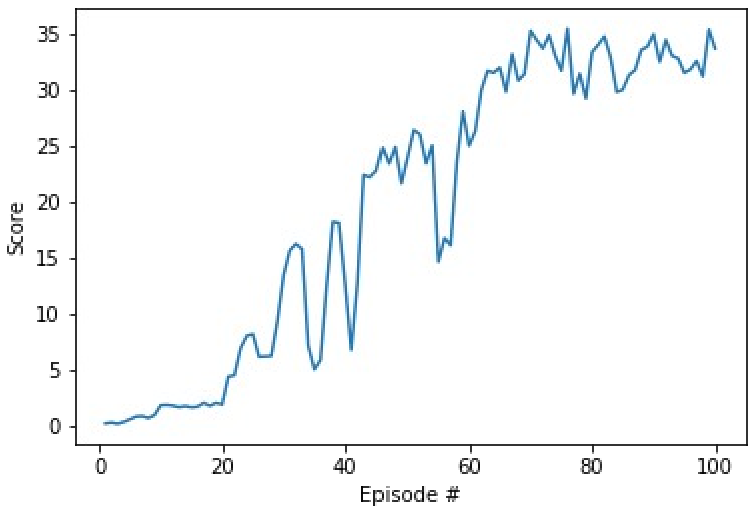
Batch Size: 64



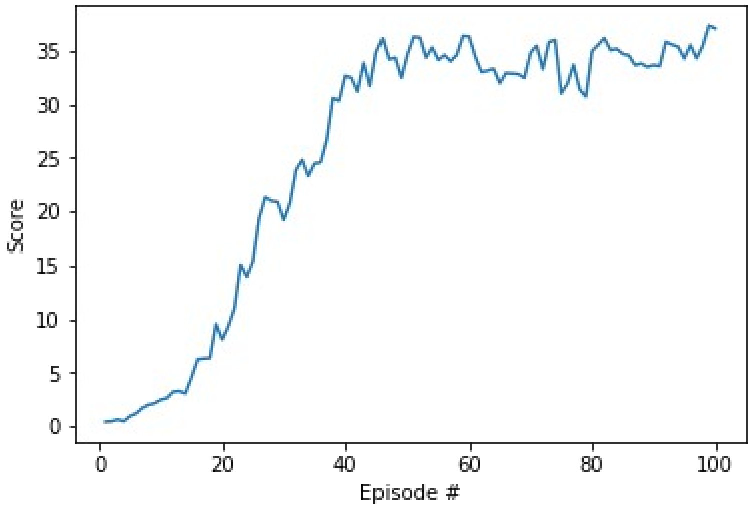
Batch Size: 128:



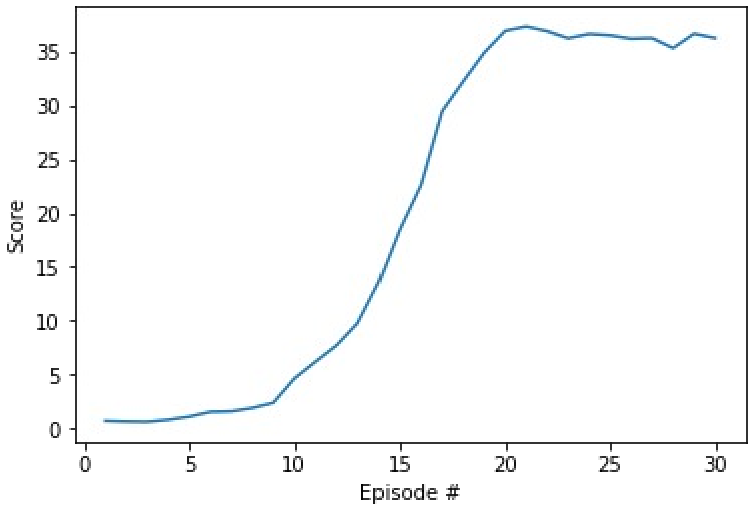
Batch Size: 256:



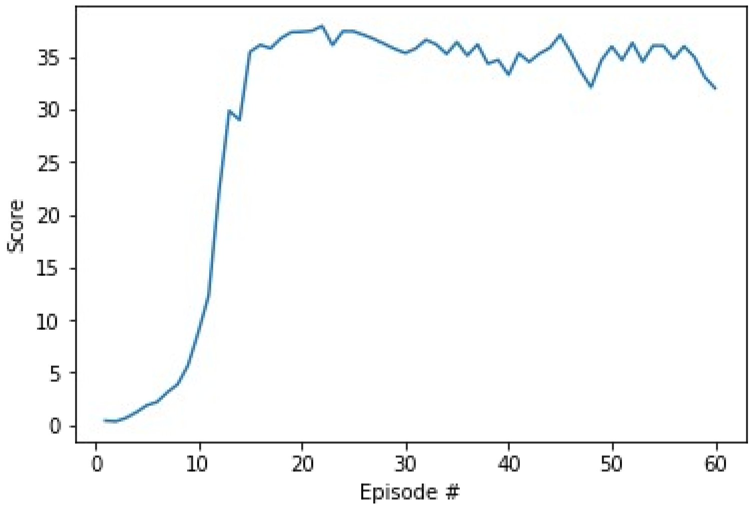
Batch Size: 512



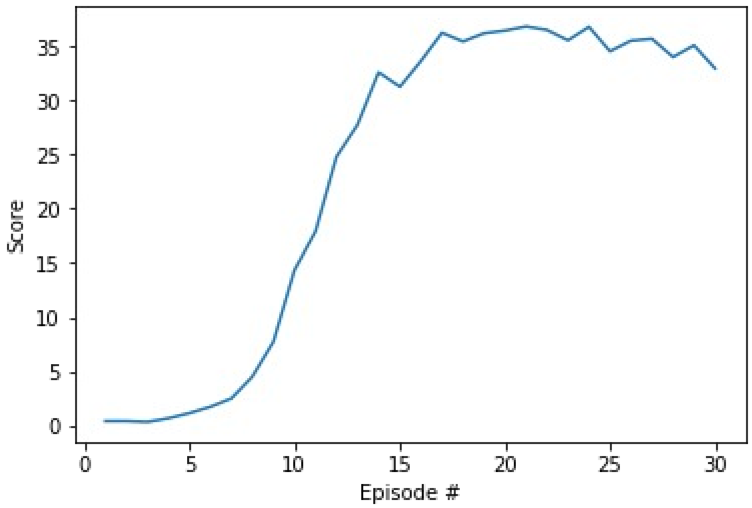
Batch Size: 1024



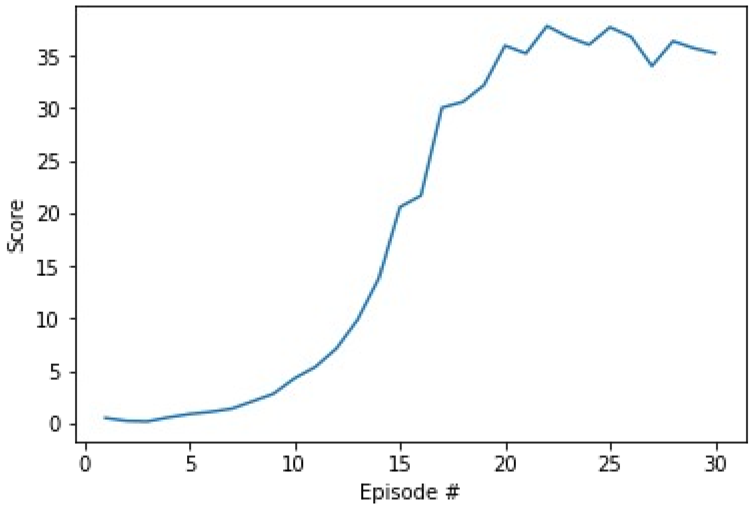
Batch Size: 2048



Batch Size: 4096

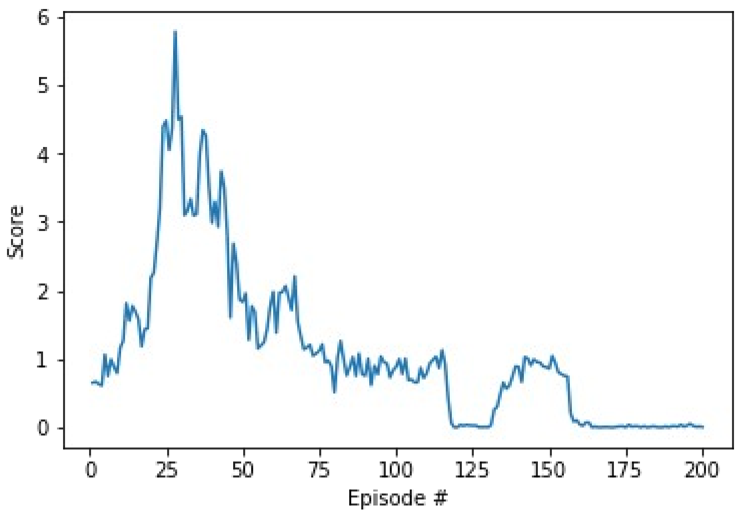


Batch Size: 8192

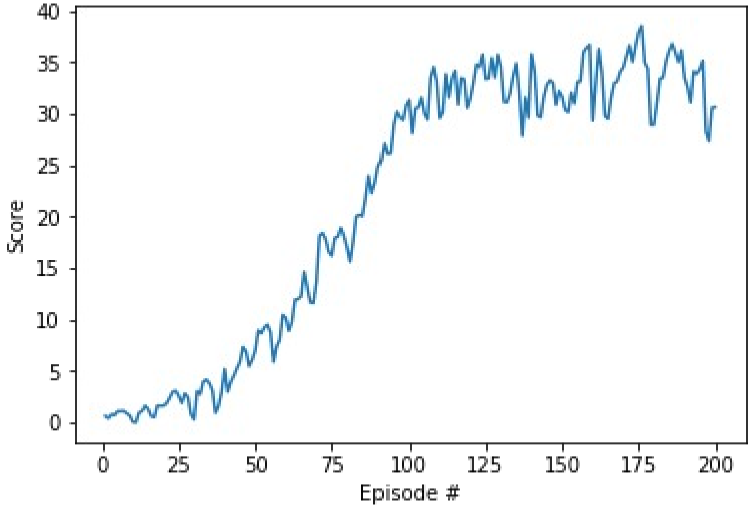


One observation is that as the batch sizes increases the number of episodes needed to get the better score is decreasing until batch size of 2048. But after that it's not that reliable and needs further study of the impacts.

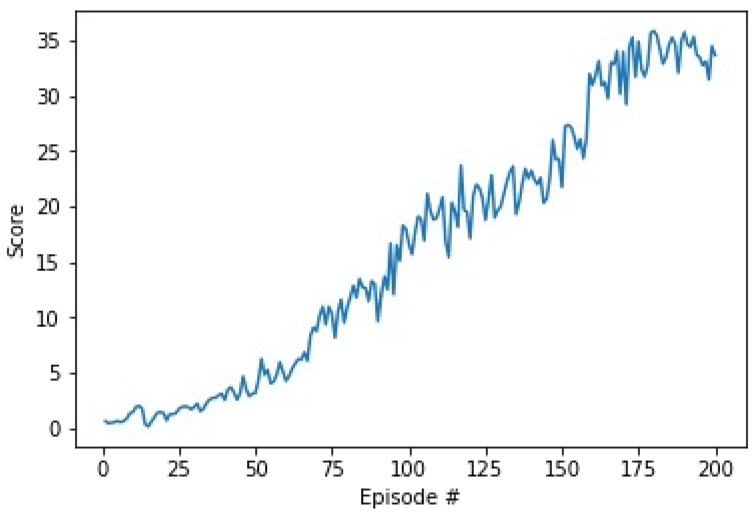
Buffer size: 1e4 = 10000, batch_size: 128



Buffer size: 1e5 = 100000, Batch Size: 128



Buffer Size: 1e6 = 1000000, Batch Size: 128



The observations related to buffer size is that there should be enough buffer to learn from otherwise convergence will not happen. And at the same time if the buffer is too large, the convergence will take long time.

The code is written in such a way that each of the hyperparameters can be arrange in a recurrent for loops and run to get the intuition!

When I tested in the notebook, my driver code is as below:

```
start = time.time()

batch_sizes = [128, 256, 512, 1024, 2048, 4096]
episode_sizes = [240, 200, 160, 120, 80, 40]

scorelist = []
batch_times = []

for batch_size, episodes in zip(batch_sizes, episode_sizes):
    loopstart = time.time()
    ddpgagent = DDPGAgent(33, BATCH_SIZE=batch_size)
    scores = ddpgrunner(ddpgagent, episodes = episodes, print_every=10)
    scorelist.append(scores)
    loopend = time.time()
    print("\n")
    print("\rBatch size: ", batch_size, " took ", loopend - loopstart, " seconds.")
    print("\n")
    batch_times.append(loopend - loopstart)

plotGraphs(scorelist, cols = 2, rows = 3)

end = time.time()
print('Elapsed ', end - start, ' seconds.')
```

Training Observations:

I trained most of time on 1080Ti initially but recently happened to get hands on an RTX Titan. Then I switched on to RTX Titan the training on RTX Titan is much faster than 1080Ti. Here is a brief comparison for a btch size of 256. We can see the training happens in less episodes and also less time.

RTX Titan log:

```
Starting training for batch size: 256...


Episode 10      Average score in the latest 10 episodes: 1.0321999769285322
Last 10 episodes took 141.58201599121094 seconds
Episode 20      Average score in the latest 20 episodes: 4.5966748972563085
Last 10 episodes took 146.64854836463928 seconds
Episode 30      Average score in the latest 30 episodes: 11.26459974821657
Last 10 episodes took 144.36819458007812 seconds
Episode 40      Average score in the latest 40 episodes: 17.024437119474168
Last 10 episodes took 145.07437419891357 seconds
Episode 50      Average score in the latest 50 episodes: 20.435709543226288
Last 10 episodes took 143.36664724349976 seconds
Episode 60      Average score in the latest 60 episodes: 22.494207830548596
Last 10 episodes took 143.63900113105774 seconds
Episode 70      Average score in the latest 70 episodes: 24.168049459801985
Last 10 episodes took 143.34466695785522 seconds
Episode 80      Average score in the latest 80 episodes: 25.371443182904038
Last 10 episodes took 143.47706651687622 seconds
Episode 90      Average score in the latest 90 episodes: 26.34437163337962
Last 10 episodes took 143.36613988876343 seconds
Episode 100     Average score in the latest 100 episodes: 27.207559391863644
Last 10 episodes took 143.4125075340271 seconds
Episode 109     Average score in the latest 100 episodes: 30.185109325310222
The score average is over 30 in the last 100 episodes. Stop the training.

Batch size:  256  took  1568.266587972641  seconds.
```

1080Ti Log:

```
Starting training for batch size: 256...


Episode 10      Average score in the latest 10 episodes:                      0.45984998972155156
Last 10 episodes took 298.4744713306427 seconds
Episode 20      Average score in the latest 20 episodes:                      1.412924968418665
Last 10 episodes took 299.30443954467773 seconds
Episode 30      Average score in the latest 30 episodes:                      3.0968832641125967
Last 10 episodes took 301.15964126586914 seconds
Episode 40      Average score in the latest 40 episodes:                      5.9201123676751735
Last 10 episodes took 305.7714216709137 seconds
Episode 50      Average score in the latest 50 episodes:                      9.823269780432804
Last 10 episodes took 304.939120054245 seconds
Episode 60      Average score in the latest 60 episodes:                      13.361841368006232
Last 10 episodes took 298.8515875339508 seconds
Episode 70      Average score in the latest 70 episodes:                      16.253028208145075
Last 10 episodes took 302.5307743549347 seconds
Episode 80      Average score in the latest 80 episodes:                      18.583974584615788
Last 10 episodes took 301.8070456981659 seconds
Episode 90      Average score in the latest 90 episodes:                      20.446743987424092
Last 10 episodes took 306.9655909538269 seconds
Episode 100     Average score in the latest 100 episodes:                      22.03210450754408
Last 10 episodes took 300.9576804637909 seconds
Episode 110     Average score in the latest 100 episodes:                      25.50928942982294
Last 10 episodes took 301.6271622180939 seconds
Episode 120     Average score in the latest 100 episodes:                      28.744569357508794
Last 10 episodes took 305.21737837791443 seconds
Episode 124     Average score in the latest 100 episodes:                    30.038074328596707
The score average is over 30 in the last 100 episodes.                    Stop the training.

Batch size:  256  took  3755.677644968033  seconds.
```