Report for Project 2 – Continuous control

The project involves using DDPG or its variants in pytorch to train a robotic arm to follow a sphere around its center.

I followed similar architecture to the one we implemented in the exercises for DDPG. I adapted the code slightly to fit the new environment.

I changed the code to work with 20 agents. This involved ensuring the data types, sizes of various data structures used were flexible for multi-agent training.

Observations

- Initially I tried following the instructions in the exercise as closely as possible. I implemented a single agent with DDPG, double DQN first. The results were terrible with max score around 1.
- After that, I extended the code to work with multiple-agents. This step took a while to execute. The score increased to 2 which was still very far away form desirable score of 30.
- After that, I attempted a lot of tricks. Some recommended in the exercise - gradient clipping when training the critic network, updating the networks 10 times after every 20 timesteps. None of these attempts improved the results.
- I then changed the actor and critic networks by making them 2-3 layers only and added batch normalization. This improved the results slightly.
- Up until here, the max score I was getting was below 3.
- After googling and looking at the forum, nothing seemed obvious to me that would improve the results significantly.
- After lots of trial and error, I change the noise in my OUNoise class to standard normal noise. It was coded (probably from previous exercise) as uniform distribution.
- I set the weight decay to be zero so that the network doesn't forget the sparse rewards.
- This immediately improved the results and I was able to get the score above 30.
- Overall, my observation is that current set of hyper parameters, noise models that works is mostly luck and can be tricky to optimize – given the short time frame. SEED parameter also played a major role in getting good results.

Learning algorithm:
ddpg_agent.py describes the agent.

I used 2 actor (local) networks and 2 critic (target) networks – one each for local and target. I also used a replay buffer and noise model – OUNoise for the agent. I used Adam optimizer. A soft update is performed using tau as the control variable. The number is relatively small, so it only updates the target by a small number each iteration.

Hyperparameters:
BUFFER_SIZE = int(1e5) # Number of episodes to keep in memory (experience replay)
UPDATE_EVERY = 1
BUFFER_SIZE = int(2e5) # replay buffer size
BATCH_SIZE = 128   # 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

Model architecture:
The model is contained in model.py.

I used 2 hidden layers network for Actor and 1 hidden layer network for Critc.
For Actor network, I used ReLU for activations and 2 sets of batch normalizations before 1st and 2nd linear transformations and ReLU activations.

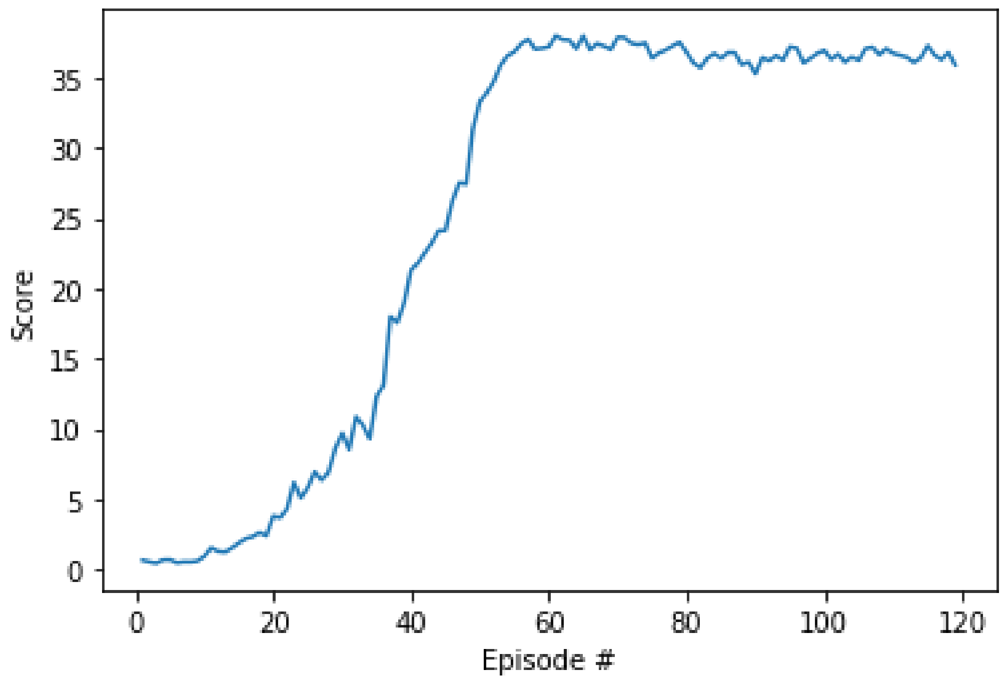
For Critic network, only one batch normalization after the first hidden layer followed by linear transformation and ReLU activation worked well for me.

Continuous_Control.ipnyb contains the python notebook to load unity environment and calls to train the agent. The walkthrough of the code is in readme file.

Results

```
Episode 100    Average Score: 23.33 Average Score: 23.33
Episode 119    ,local score 35.92     , Average Score: 30.05
```

The plot of rewards per episode is shown below:



Future work:

We can try other algorithms such as SAC, A3C, A2C, PPO etc. to improve the results. More effort can be put in fine tuning the hyper parameters – learning rate for actor, critic; number of agents, epochs, iterations, weight_decay etc to improve the results based on current DDPG implementation.