Deep Reinforcement Learning Nanodegree

Project 2 : Continuous Control

Introduction

In this problem, we were required to build an agent to control a double jointed robotic arm that can maintain its contact with green spheres as long as possible.

A reward of +0.1 is provided for each step that the agent's hand is in the goal location.

I worked with a 20 agent version of the Reacher Environment , where, at a given time, 20 state vectors were generated by 20 agents acting independently.

State space : The vector observation was 33 dimensional input including position, rotation, angular velocity and other kinds of information.

Action Space : The agent can take 4 different actions in continuous state space (-1,1), each corresponding to torque applied to the joints.

Goal : The goal is to obtain a average reward of 30 over a course of 100 consecutive episodes agent interacts with the environment

Learning Algorithm

We are using a popular variant of actor-critic method, DDPG algorithm to train the agent. These algorithms address shortcomings of both policy based and value based methods.

It leverages policy based methods to take best actions that maximize agents' total cumulative reward over a trajectory, while also utilizing the value based methods idea to effectively estimate goodness different action values. And the key thing is, the algorithm is learning policies in high dimensional continuous action spaces.

More specifically, the policy network (actor) is updated using deterministic policy gradient while the action value estimator is updated with TD error.

Algorithm Features accompanying stable learning

Significant problem with Deep RL lies in making the gradient update (both ascent and descent) stable, since we are just working with sample interactions. Due to the high variance of environment signal, we leverage certain techniques to assist in stable learning as well as continual exploration, since, unlike DQN, we don't have privilege to use epsilon greedy over discrete actions.

1) **Ornstein-Uhlenbeck noise** (Promotes exploration, while subtly tending to stay in same direction, rather than, negative actions canceling out in long run as sampled with uniform random distribution)
2) **Batch Normalization** (By adding Batch normalization to every layer, motivated stable gradient update.)

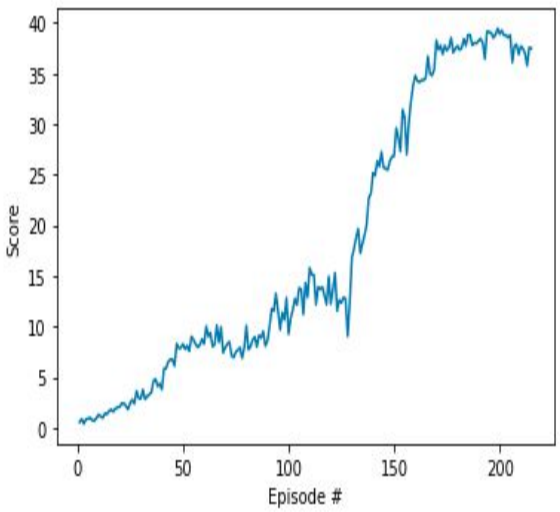


Fig 1 . Adding Batch Norm to First Layer

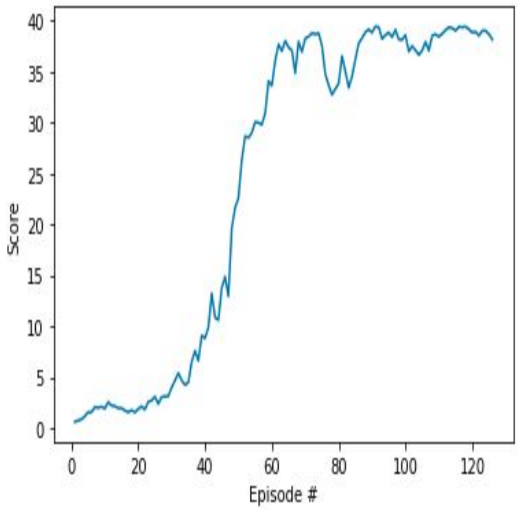


Fig 2. Only After Adding Batch norm to each layer

Figure above makes it more evident the key role played by batch normalization as mentioned in DDPG paper as well.

3) **Fixed Targets** (Same motivation as in DQN)
4) **Soft Updates** (Defined by parameter *tau* that slowly mixes local net parameters into target)
5) **Experience Replay** (Motivated by DQN to break sequential dependency)

Neural Network Architecture

Both actor and critic are fundamental two layer feed forward neural networks. Additional detail worth mentioning is, how addition of batch normalization to each layer of Actor, rather than just the first one, accelerated the learning process. But I have noticed, in deeper nets, adding batch norm to each layer acts more like an overkill and thus, only using the first few layers will do the job.

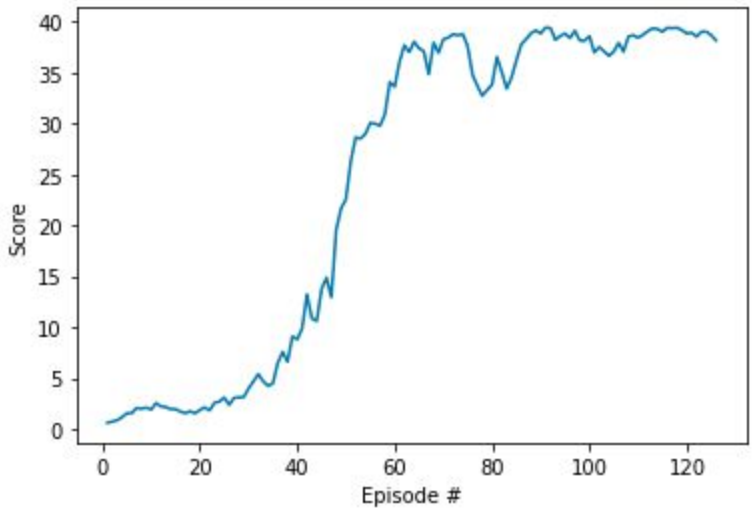
Hyperparameters

```
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

## for this problem,  these values accelerated the learning process
LR_ACTOR = 1.2e-3         # learning rate of the actor
LR_CRITIC = 1.2e-3      # learning rate of the critic

## As per suggestion in the course lectures
UPDATE_INTERVAL = 20       # how often to update the network
NET_UPDATE = 10          # how many times to update the network
```

Plot of Rewards



Ideas for Future work

- Using mentioned algorithms like PPO, TRPO, D4PG as mentioned, which are speculated to be more robust in continuous action spaces. Being familiar with PPO, It's intuitive on how problems like this are better suited to maximizing over the trajectory.

- Combining Prioritized Experience Replay for better sampling during Critic's update.