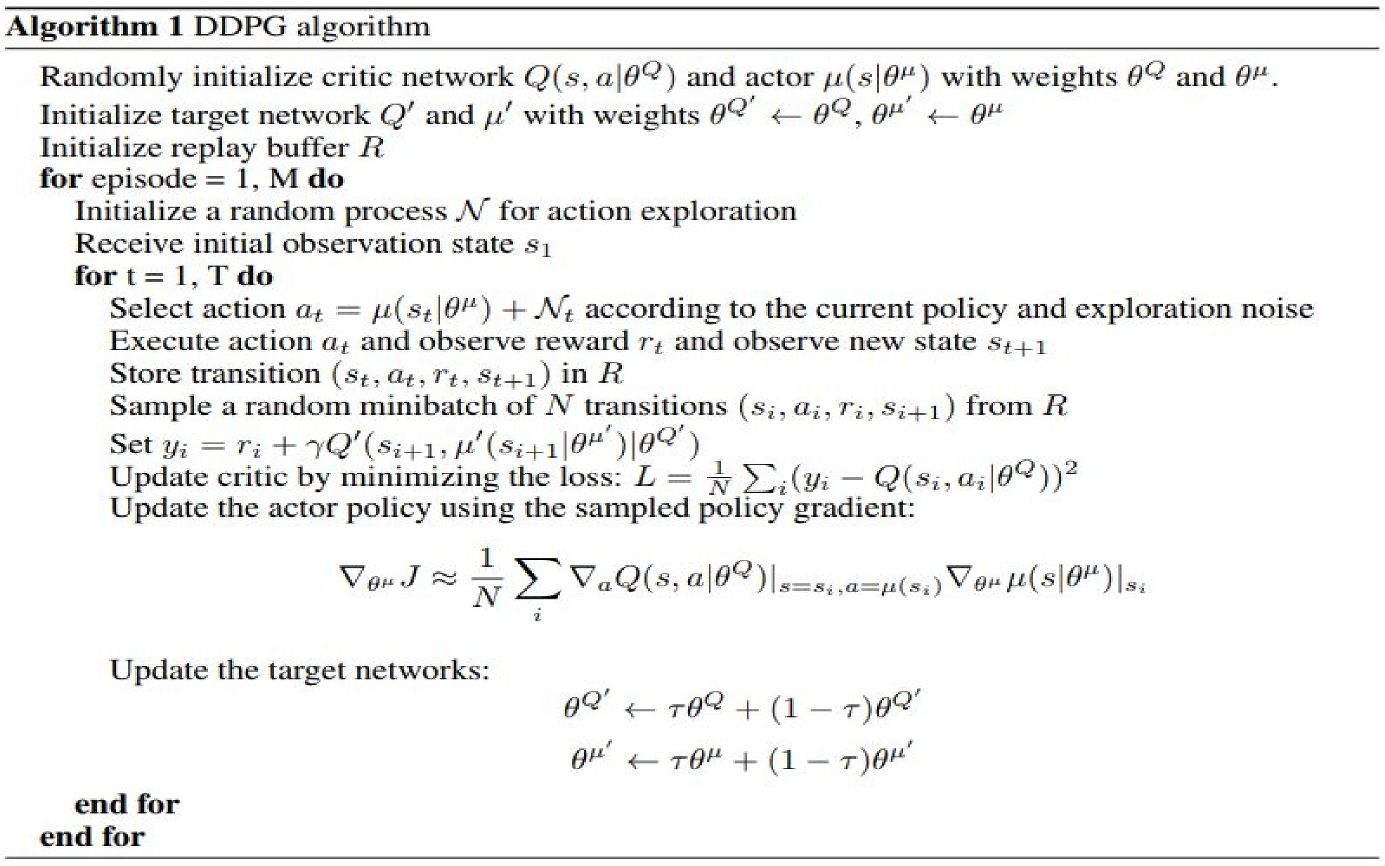
Continuous Control Project Report

Solution

This project solved a complex environment with continuous action spaces with multiple agent.

- The task solved refers to a continuous control problem where the agent must be able to reach and go along with a moving ball controlling its arms.
- The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints.
- The reward of +0.1 is provided for each step that the agent's hand is in the goal location. The goal is to get an average score of +30 over 100 consecutive episodes.

I'm using the DDPG algorithm to solve this problem. DDPG is a model-free policy based learning algorithm in which the agent will learn directly from the un-processed observation spaces without knowing the domain dynamic information. DDPG employs Actor-Critic model in which the Critic model learns the value function like DQN and uses it to determine how the Actor's policy based model should change. The Actor brings the advantage of learning in continuous actions space without the need for extra layer of optimization procedures required in a value based function while the Critic supplies the Actor with knowledge of the performance.



Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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
```

As there are 20 arms there can be one brain for each arm or one brain for all arms. I use one one single “brain” instead of 20 individual brains as training one brain will definitely be quicker than 20 brains.

Model Architecture

The Actor model is a neural network with 2 hidden layers, Tanh is used in the final layer that maps states to actions. Batch normalization is used for mini batch training.

The Critic model is with 2 hidden layers,with the final layer is a fully connected layer that maps states and actions to Q-values

For the neural models:

- Actor
    - Hidden: (input, 128) - ReLU
    - Hidden: (128, 64) - ReLU
    - Output: (64, 4) - TanH
- Critic
    - Hidden: (input, 128) - ReLU
    - Hidden: (128+action_size,64) - ReLU
    - Output: (64, 1) - Linear

Hyperparameters

Followings are hyper-parameter settings,

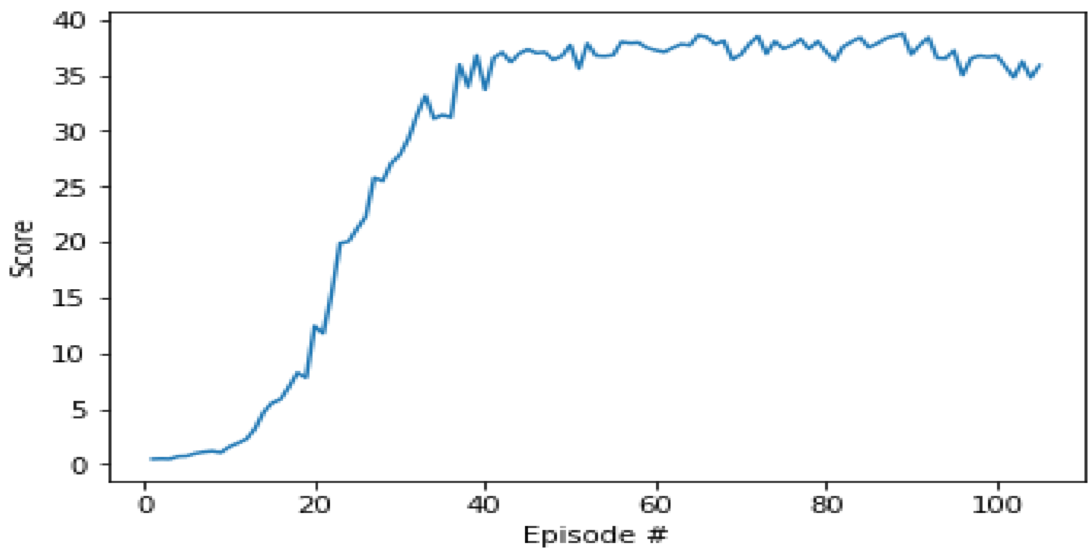
- Learning Rate (Actor/Critic): 1e-3
- Weight Decay: 0.0000 # L2 weight decay
- Batch Size: 1024
- Buffer Size: 1000000
- Gamma: 0.99 # reward discount factor
- Tau: 1e-3 # soft update for parameters in target network
- Ornstein-Uhlenbeck noise parameters (0.15 theta and 0.2 sigma.)
- Hidden Layer 1 Size: 128
- Hidden Layer 2 Size: 64

The Ornstein-Uhlenbeck noise is added in action values for action exploration. The Ornstein-Uhlenbeck process adds a certain amount of noise to the action values at each timestep. This noise is correlated to previous noise, and therefore tends to stay in the same direction for longer durations without canceling itself out. This allows the arm to maintain velocity and explore the action space with more continuity.

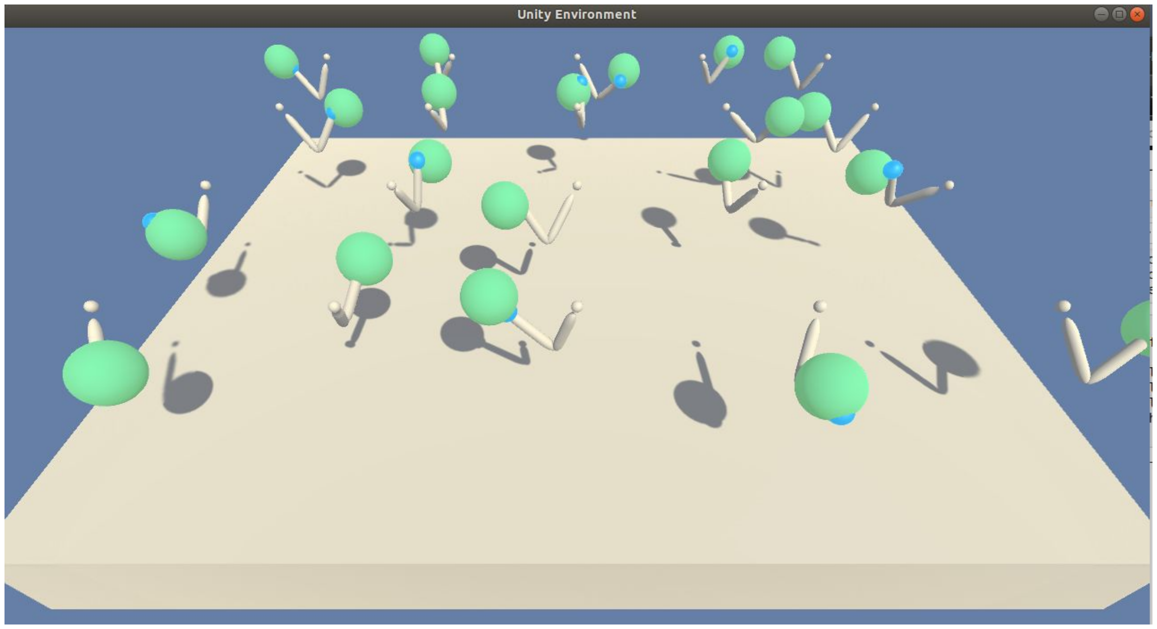
Final Results

This implementation which solved the environment (average scoresis at least +30.) The console output and a plot of the scores are displayed below.

Problem Solved after 105 epsisodes!! Total Average score: 30.28



This shows agents works well after load the saved model to actor and critic



Future Improvements

- Prioritized experience replay — rather than selecting experience tuples randomly, prioritized replay selects experiences based on a priority value that is correlated with the magnitude of error. This can improve learning by increasing the probability that rare and important experience vectors are sampled.
- Try with PPO or A3C that use multiple copies of same agent to run distributed environment.