Project 2: Continuous Control

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

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. Every entry in the action vector should be a number between -1 and 1.

Implementation

The implementation is utilizing DDPG (Deep Deterministic Policy Gradients) with 2 hidden layers (300, 200 nodes) for both actor & critic networks. Code is adapted from Udacity's ddpg-pendulum repository.

Hyperparameters chosen for the implementation are below:

```
SEED = 0                      # random seed for python, numpy & torch
episodes = 2000               # max episodes to run
max_t = 1000                  # max steps in episode
solved_threshold = 30         # finish training when avg. score in 100 episodes crosses this threshold

batch_size = 128              # minibatch size
buffer_capacity = int(1e6)    # replay buffer size

learn_every_n = 20            # how many steps to collect experiences before learning
learn_updates = 10            # how many times to take samples from memory while learning
gamma = 0.99                  # discount factor
tau = 1e-3                    # for soft update of target parameters
learning_rate = 1e-4          # learning rate for both actor & critic networks
max_norm = 1                  # clipping of gradients to prevent gradient explosion
```

Learning Algorithm

Algorithm that was used in this work for solving the enviromnent is described in paper: CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING, Timothy P. Lillicrap et al.

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
```

DDPG is an Actor-Critic method that uses value function and direct policy approximation at the same time. There are two internal types of neural networks:

- Actor network - transforms state to action values. In this environment there are 4 action values.
- Critic network - transforms state and action values to a quality measure of this state (Q(s, a))

DDPG is using both value function (Critic network) & policy approximation (Actor network) because using only one kind of approximation we get:

- policy based methods - have high variance
- value function methods - have high bias

Moreover, in contrast to DQN we can use continuous action space.

Similarly like for DQN we utilize

Experience Replay

A buffer with experience tuples (s, a, r, s'): (state, action, reward, next_state)

Q-targets fixing

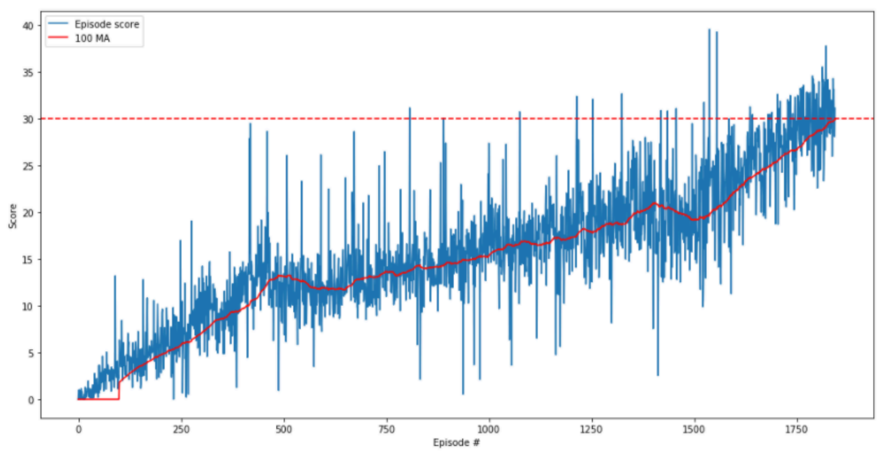
We create 2 neural networks (NN): local and target. Then fix target NN weights for some learning steps to decouple the local NN from target NN parameters making learning more stable and less likely to diverge or fall into oscillations.

Hence in reality we have to have 4 neural networks:

- Critic target NN
- Critic local NN (for execution)
- Actor target NN
- Actor local NN (for execution)

Rewards

As shown below agent learned the environment fairly quickly. In around 500 episodes it reached average 13+ reward.



Ideas for future works

Prioritized Experience Replay

This approach comes from the idea that we want to focus our training on the actions that were "way off" what we did. That means the higher the TD error the higher priority. We store in the experience buffer the probability of choosing the experience tuple depending on the TD error. Then we sample the experiences based on this probability. There is one caveat that it's required to add small epsilon to the probability (*p*) since setting the tuple's p = 0 will make it practically disappear and the agent will never see that tuple again.

Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG)

In the trust region, we determine the maximum step size that we want to explore for optimization and then we locate the optimal point within this trust region. To control the learning speed better, we can be expanded or shrink this trust region in runtime according to the curvature of the surface. This technique is used because traditional policy based & gradient optimization model for RL might make a too large step and actually fall down (in terms of rewards) and never recover again.

Try the same algorithms on Crawler environment