Project: Continuous Control with DDPG

Sang-Ho Kim

This is the report submitted for the Udacity project “Continuous Control”. This project is to solve the Reacher environment using DDPG (Deep Deterministic Policy Gradient). In Reacher 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 the 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.

The environment contains 20 identical agents, each with its own copy of the environment. The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30.

Learning Algorithm

The solution was started by modifying ddpg-pendulum solution in the Udacity GitHub repo of deep-reinforcement-learning: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum. As there are 20 agents, while the original was written for 1 agent, some modifications were made: (1) the scores were saved with average value, (2) the input arguments in OUNoise were modified to take the tuple of (number of agents and action size), (3) the OUNoise class save experiences for each agent in the namedtuple. It contains ddpg agent (ddpg_agent.py) and deep neural network model (model.py). The file ddpg_agent.py contains the classes of Agent, OUNoise for Ornstein-Uhlenbeck process), ReplayBuffer for storing experience tuples. The file model.py contains Actor (policy) and Critic (value) models inheriting from torch.nn. Both Actor and Critic model have 1 input layer, 1 hidden layer and 1 output layer. The number of nodes in the hidden layer were slightly changed from the original values. DDPG learns a Q-function using off-policy data and Bellman equation and a policy using the Q-function.

In DDPG, the optimal action-value function defined by

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

is used to define the mean squared Bellman error (MSBE) function,

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \right]$$

DDPG is to minimize the MSBE loss function.

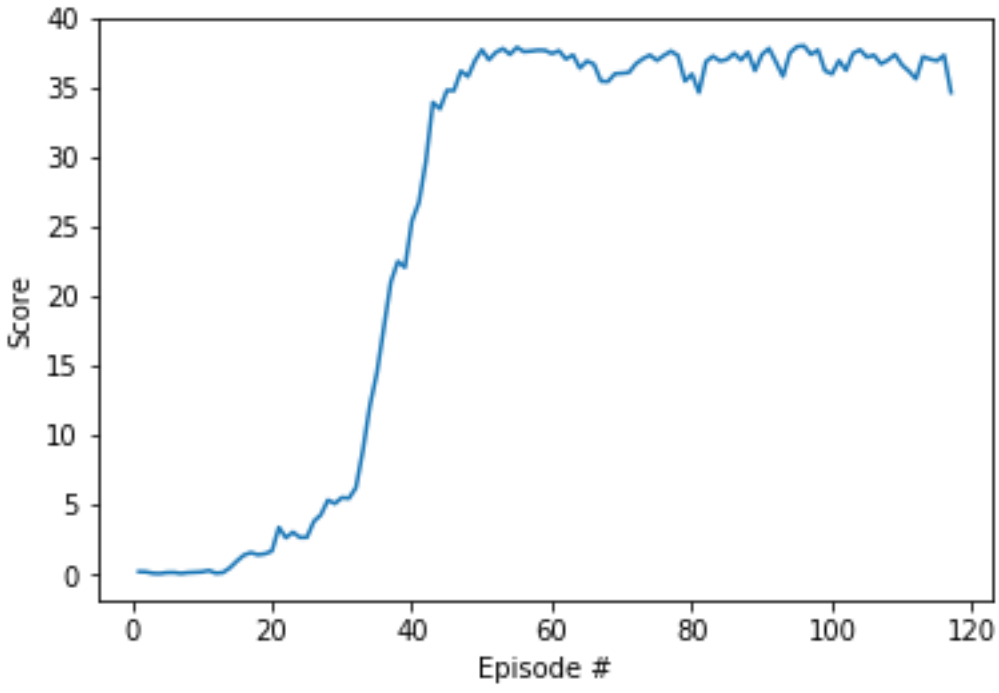
In the solution, the following hyper parameter values were used:

```
maximum time step for each episode - long enough until any state done is true
mu, theta, sigma in OUNoise = 0., 0.15, 0.2
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
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-4        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```

The deep learning model in both Actor and Critic contains input layer of (state size x 256 nodes), 1 hidden layer of (256 + action size) x 128, and 1 output layer (128 x 1). The first 2 layers in Actor contains the relu activation function and tanh in the output layer. The Critic takes state as the input, which is fed to the hidden layer and concatenated with action obtained from Actor as the input.

Plot of Rewards

The following rewards plot was obtained. In the beginning, the model did not learn much until about 15 episodes and it suddenly kicked up above the score of 35. It seems partly because the learning rates were set as low as 1e-4 for both actor and critic. Once the score went over 35, it went steadily flat, implying a good stability.



Ideas for Future Work

It is quite exciting to learn that DDPG works very well with a continuous control problem. It will be also interesting to solve different continuous control problems using the same solution. I will try Crawl as well starting with the same hyper parameters see if it works. If not, I will play around the hyper parameters to obtain a good performance.