Continuous Control Project

Udacity Nanodegree on Deep Reinforcement Learning.

By Peerajak Witoonchart

Project Rubics:

- The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.

- A plot of rewards per episode is included to illustrate that the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30.

- The submission reports the number of episodes needed to solve the environment.

- Idea for future work: The submission has concrete future ideas for improving the agent's performance.

Problem:

In this project, we wish to put the robot arm into a moving sphere target. The difficulty is that this is a continuous control problem, where the action variable is a vector of real number instead of discrete variables as we have done in previous work.

Idea to the solution:

In the last project, we had quite similar difficulty. We used to have discrete value Q table, but was challenge with continuous space. We use Neural Network to represent the Q table. Yet the previous project was about discrete action space. In this project, we are challenged by continuous action space. We, once again, use Neural Network to represent action selection. However, now that we have 2 neural networks, how could we learn them together? This two neural networks fit well into Action-Critic Framework.

To explain how this algorithm works, let start with Q- Learning in P1 problem.
Here is Q-Learning summary
- Store Samples
    1. - Observe the current state S.
    2. - Choose Action A from epsilon greedy
    3. - Observe Reward R, and next state (Snew)
    4. - Store a tuple (S,A,R,Snew) to the buffer
- Learning
    1. - Obtain random tuple from the buffer
    2. - set Target y_i = r_i + gamma max_a target_network(S,a) where a is a iterator over all possible action value in Action space

3. - update with gradient ascent on local_network the difference between y_i, and the local_network prediction local_network(S,A)
4. - Once a while copy weights of local_network to target_network

Back to our DDPG continuous control problem, our idea is to replace the way we choose Action a from epsilon greedy to Action Prediction. In our case, the action space is now [-1,1]^4, we therefore, model action predictor with neural network whose input_size is the dimension of the input state, and the output size is the action space dimension.

We need critic network to associate our models to rewards. As with Q Learning, we need local_network,and target_network for critic network.

How can we learn (back propagation) the Action network, and Critic network? Notice that the Q-network Learning

y_i = r_i + gamma max_a target_network(S,a)?

We choose change this update like this

y_i = r_i + gamma critic_target_network(S,actor_target_network(S)).

In other words, we hope that actor_network(S) would predict the value of a* which maximize max_a target_network(S,a) . **Now since Critic_local_network should be the predictor of y_i, we learn the mean square error between y_i, and the critic_local_network, and back propagate down the line.** Once awhile, we copy the critic's local network parameters to target network parameters just like Q Learning do. For the action network, we use update rule of Q-update to update action_local_network.

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned} \tag{6}$$

By substitute critic_local_network with the action predicted by action_local_network, and do back propagation for gradient ascent, we could learn the action network. Once awhile, we copy the action's local network parameters to target network parameters just like Q Learning do.

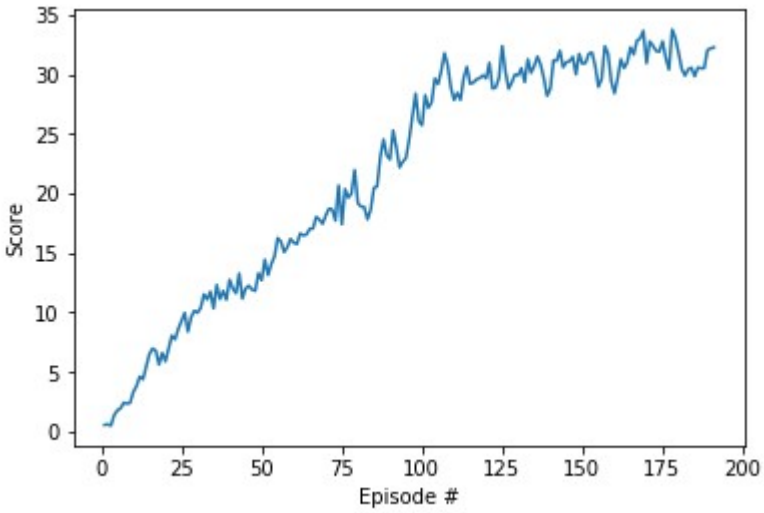
Special attention must be paid for a "done" state. In done state, the y_i equals r_i.

Experiment

After many unsuccessful parameters, finally, this set of hyper parameters do the job.

BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 512 # minibatch size
GAMMA = 0.99 # discount factor
TAU = 1e-3 # for soft update of target parameters
LR_ACTOR = 5e-5 # learning rate of the actor
LR_CRITIC = 5e-5 # learning rate of the critic
WEIGHT_DECAY = 0.0 # L2 weight decay
LEARN_N_INTERVAL = 20
LEARN_N_TIMES = 10

The result score is score of each episode vs number of training episode. The score is an average of multiple agent scores. The training is finished at episode 191 where average score over previous 100 episodes is greater than 30.



Idea for future work.

I have been trying a lot of time working on pure PPO algorithm over DDPG. The DDPG success must have been based on its critic-action framework. I would like to know it this is true. In otherwords, my future work should be to carefully using scientific method to analyze the effect of critic-action framework on DDPG.