HW4 Report

1st Yifan Wu
yiw084@ucsd.edu

I. Setups

A. Replay Buffer

The replay buffer is implemented by fixed size numpy arrays. It is a FIFO ring buffer which stores state, action, reward, next state, and done. 1,000 samples with state-transitions are sample uniformly from action space and stored in this buffer before training start.

B. Actor and Critic Networks

Both actor and critic are implemented as neural networks. The actor network is consist of two fully connected hidden layer with size 400 and 300 respectfully. We use ReLU as the hidden activation and the wrap the output with Tanh to bound it between $[-1, 1]$.

The critic networks consists of two fully connected hidden layer size of 402 and 300 and two ReLU activation. There is an identity output activation. Extra 2 dimension is the action dimension, since it is introduced to the network from the second layer. The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-3\times10^3, 3\times10^3]$. All other layers are initialized from uniform distributions $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$, where f is the fan-in of the layer.

This is exactly the same as proposed in original DDPG paper.

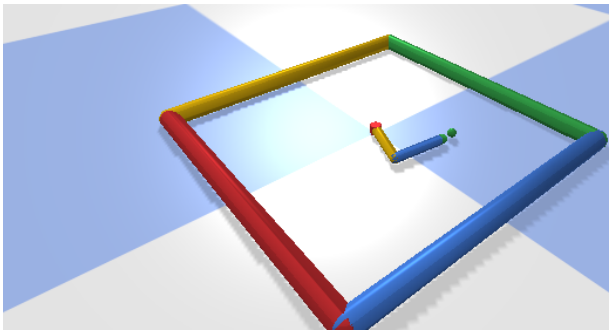

Fig. 1. *Pybullet Reacher Env*

II. Results

Three DDPG agent were trained using different seed: 0, 1, 7 respectively. The result are shown in the following figure. We evaluate there policy at the end of each epoch with a same testing environment which has same seed 42

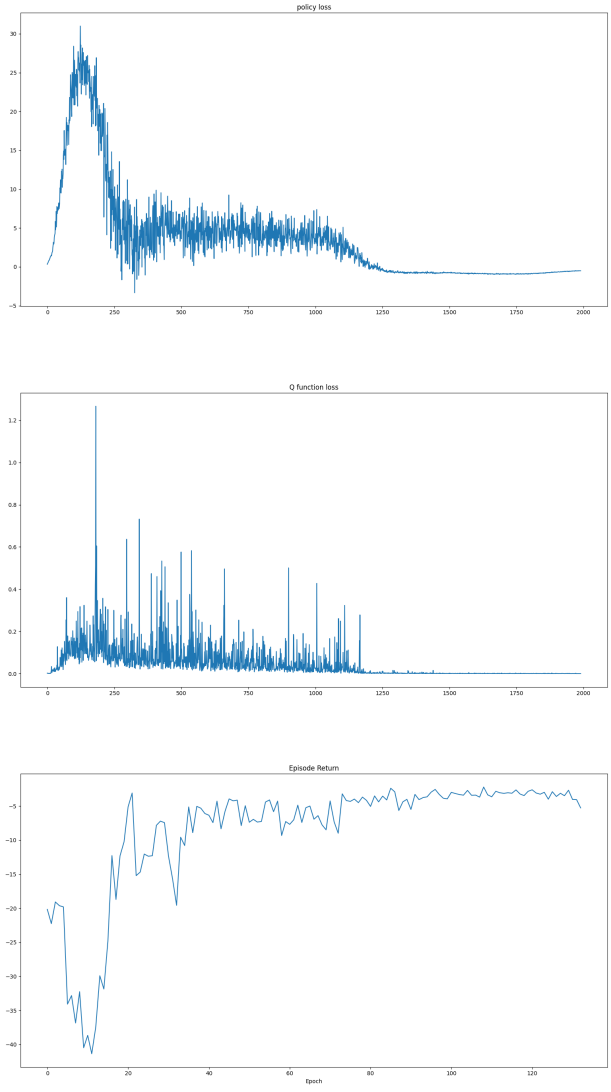

Fig. 2. *DDPG Policy Loss, Q-function Loss and Episode Return with seed 0*

Fig. 3. Policy Loss, Q-function Loss and Episode Return with seed 1

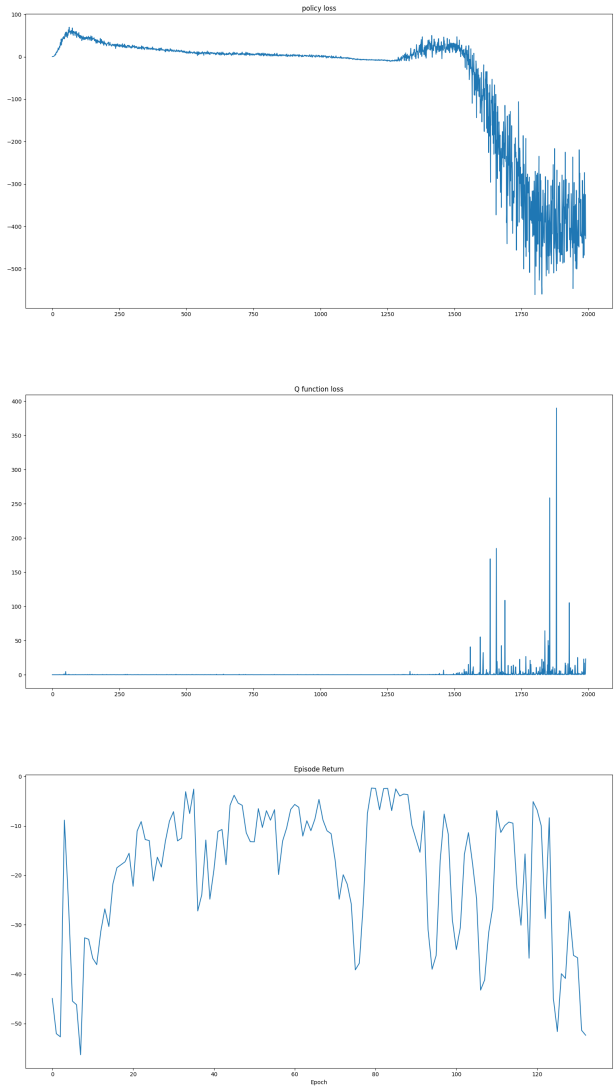

Fig. 4. *DDPG Policy Loss, Q-function Loss and Episode Return with seed 7*

III. Discussion

In assignment 3, we using the vanilla policy gradient. The vpg agent is able to converge in $2e5$ steps. We used a batch size 2,000 and 200 iteration. Since vpg is an on-policy algorithm. The data have to be collected freshly. We need $1,000 \times 200 = 400,000$ samples. However, in DDPG, we uses $\frac{totalsteps}{epochperstep} \times batchsize + Startsteps = \frac{2e5}{15000} \times 128 + 1,000 = 17024$ samples and achieves same results as vpg. Therefore, DDPG is more sample efficient than vpg. The downside of DDPG is the learning speed. DDPG is a variant of Q Learning which is typically slow. The update between current actor and critic and target actor and critic is using polyak update. This can be unstable if soft update coefficient is large. Therefore, we need slowly improve the policy and Q function with a small update coefficient.

Although, DDPG is able to achieve good performance, it is brittle with respect to hyperparameters and other kinds of tuning. As we can see from the training figure above, the learning curve changes dramatically and the performance is not guaranteed. The learned Q-function tends dramatically overestimate Q-values, which then leads to the policy breaking.

IV. Hyper-parameters

TABLE I
HYPER-PARAMETERS

Parameter	Value	Description
Env	Reacher	The environment to interact with
Gamma	0.99	Discount factor
Tau	0.005	Soft update coefficient ("Polyak update" $\in [0, 1]$)
Action noise std	0.1	Standard deviation of normal distribution action noise
Policy learning rate	1e-4	Learning rate of actor
Q function learning rate	3e-4	Learning rate of critic
Total time steps	2e5	The total number of env step) to train on
Batch size	1e3	Mini batch size
Start steps	1e3	Number of steps to collect transitions before learning starts
Step per epoch	1500	Number of steps in on epoch
train freq	150	Update the model every trainfreq steps