

P2 continuous control report

Yifan Tian

1. Learning Algorithm

The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s, a)$ is learned using the Bellman equation as in Q-learning. The actor is updated by following the applying the chain rule to the expected return from the start distribution J with respect to the actor parameters.

Below shows the pseudocode

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .

Initialize target network Q' and μ' 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

The replay buffer is a finite sized cache R . Transitions were sampled from the environment according to the exploration policy and the tuple $(st, at, rt, st+1)$ was stored in the replay buffer. When the replay buffer was full the oldest samples were discarded. At each timestep the actor and critic are updated by sampling a minibatch uniformly from the buffer. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.

A replay buffer of size 100000 was used from which was drawn random batches of

128 experiences during training. Soft updates between both actor and critic's local and target networks both mixed in a fraction $\tau=0.001$ of the local network.

HyperParameter

```
_batch_size = 128
_buffer_size = int(1e5)
_gamma = 0.99
_lr_actor = 1e-3
_lr_critic = 1e-3
_tau = 1e-3
_noise_decay = 0.999
```

Model architecture

The architecture of the network is following

Actor network layers:

```
ModuleList(
  (0): Linear(in_features=33, out_features=256, bias=True)
  (1): Linear(in_features=256, out_features=256, bias=True)
  (2): Linear(in_features=256, out_features=4, bias=True)
)
2 hidden layers with relu activation function
```

Critic network layers:

```
ModuleList(
  (0): Linear(in_features=33, out_features=256, bias=True)
  (1): Linear(in_features=256, out_features=256, bias=True)
  (2): Linear(in_features=256, out_features=1, bias=True)
)
Also 2 hidden layers with relu activation function
```

In the implementation, there are two networks, local and target.

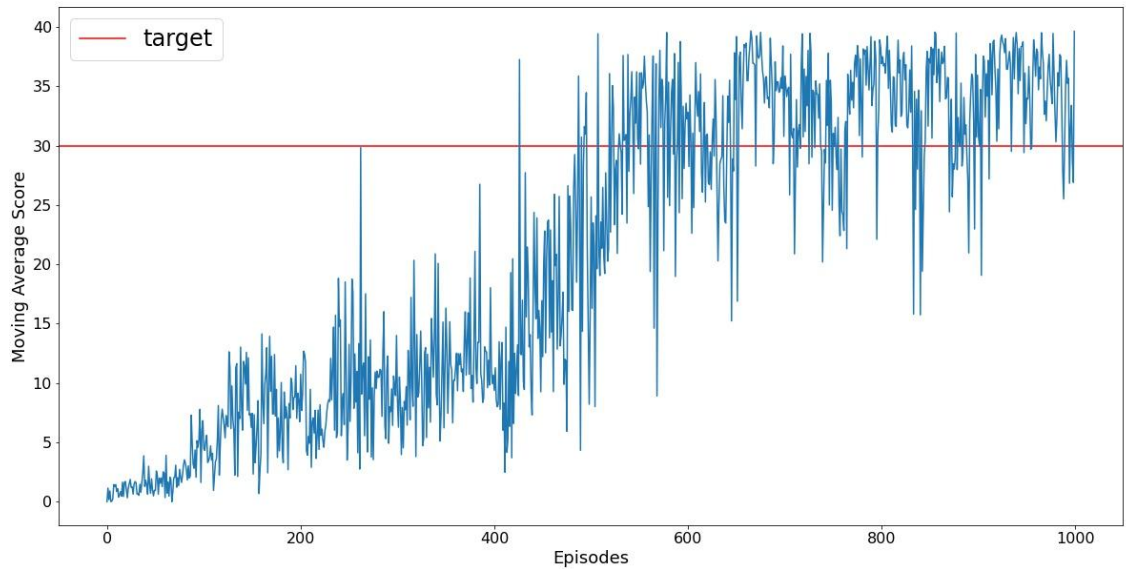
We update the local network by the following equation:

$$\theta_{\text{target}} = \tau \theta_{\text{local}} + (1 - \tau) \theta_{\text{target}}$$

We train and optimize the local network, we update the target network from the above soft update scheme. There are two networks because we want to slowly train the target network. Rather than update the network completely after each step.

2. Plot of Rewards

The following figure shows the performance of the training of DDPG agents. The red line shows the target. And after about 500 episodes. The agent can achieve a score that is better than the target consistently.



3. Ideas for Future Work

- Current training takes a long time. So Prioritized Experience Replay can be used to speed up the training. The learning rate and the use of decay noise term can also be used to speed up the training.
- By tuning the network architecture, we can improve the score. So I think there is space for improvement in model architecture.