Paper Report
Continuous Control With Deep Reinforcement Learning (DDPG) [1] *

Yilu Peng
Tandon School of Engineering
New York University
Brooklyn, New York
yp1232@nyu.edu

***Abstract*—With the limitation of DQN [2] on only discrete action domain, the DDPG paper generalizes the Deep Q-Learning algorithm to the continuous action domain.**

***Index Terms*—DDPG, Deep Q-Networks, Deterministic Policy Gradient**

I. Summary

This paper introduced a new algorithm called DDPG (Deep Deterministic Policy Gradient). It presents a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces.

An obvious approach to adapting deep reinforcement learning methods such as DQN to continuous domains is to to simply discretize the action space. However, the dimensionality may become very large if we do a coarse discretization. Such large action spaces are difficult to explore efficiently, and naive discretization of action spaces needlessly throws away information about the structure of the action domain, which may be essential for solving many problems.

This paper combines the work of DPG (Deterministic Policy Gradient) and DQN (Deep Q-Networks). DPG is a deterministic policy gradient algorithm for reinforcement learning with continuous actions. From DQN, DDPG adopted the idea of memory replay and different networks for actor and target. Compared with DPG, DDPG uses neural networks as function approximator.

DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network.

Parameters:

θ^Q : Q network

θ^μ : Deterministic policy function

$\theta^{Q'}$: target Q network

$\theta^{\mu'}$: target policy network

Fig. 1. Parameters

II. DDPG algorithm

A. Actor Critic

The DPG algorithm maintains a parameterized actor function $\mu\left(s|\theta^\mu\right)$ 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.

$$Q^\mu\left(s_t,a_t\right)=\mathbb{E}_{r_t,s_{t+1}\sim E}\left[r\left(s_t,a_t\right)+\gamma Q^\mu\left(s_{t+1},\mu\left(s_{t+1}\right)\right)\right] \tag{1}$$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with random weight 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 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 $y_i=r_i+\gamma Q'(s_{t+1},\mu'(s_{i+1}|\theta^{\mu'}|\theta^{Q'}))$;
    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 by 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

Fig. 2. Algorithm

Q' and μ' are created by copying the actor and critic networks respectively, so that we can train the critic without divergence.

As shown in Fig.3, DDPG maintains an actor network and a critic network. The actor network $\mu\left(s|\theta^\mu\right)$ maps states to actions where θ^μ is the set of actor network parameters, and the critic network $Q\left(s,a|\theta^Q\right)$ outputs the value of action under that state, where θ^Q is the set of critic network parameters. To explore better actions, a noise is added to the output of the actor network, which is sampled from a random process $\mathcal{N}$.

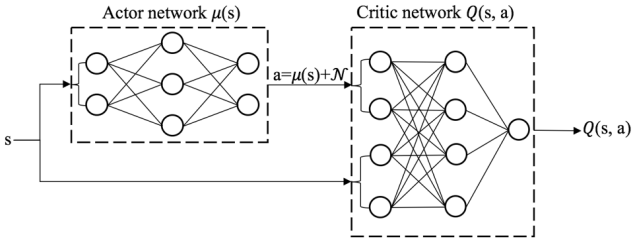

Fig. 3. Learning Architecture

B. Experience Replay Buffer

Similar to DQN, DDPG uses an experience replay buffer R to store transitions and update the model, and can effectively reduce the correlation between experience samples.

At each time, the DDPG agent takes an action a_t for s_t and receives a reward based on the new state. The transition (s_i, a_i, r_i, s_{i+1}) is then stored in replay buffer R.

The critic network is then updated by minimizing the expected difference $L\left(\theta^Q\right)$ between outputs of the target critic network Q' and the critic network Q.

$$L\left(\theta^Q\right) = \mathbb{E}\left[\left(r_t + \gamma Q'\left(s_{t+1}, \mu\left(s_{t+1}|\theta^\mu\right)|\theta^{Q'}\right) - Q\left(s_t, a_t|\theta^Q\right)\right)^2\right] \tag{2}$$

The parameters θ^μ of the actor network is the gradient of J as follows:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_{\theta^\mu} Q\left(s, a|\theta^Q\right)\big|_{a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_a Q\left(s, a|\theta^Q\right)\big|_{a=\mu(s_t)} \nabla_{\theta_\mu}\mu\left(s|\theta^\mu\right)\big|_{s=s_t}\right]\end{aligned} \tag{3}$$

C. Soft Target Updates

After the critic network and the actor network are updated by the transitions from the experience buffer, the target actor network and the target critic network are updated as follows:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{4}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{5}$$

τ denotes learning rate. $\tau << 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning.

III. Value of its work

DDPG combined DQN and DPG to handle the task for continuous action domain.

This algorithm consists of three key components: (i) actor-critic framework that models large state and action spaces; (ii) target network that stabilizes the training process; (iii) experience replay that removes the correlation between samples and increases the usage of data; (iv) soft target updates that improves the stability of learning.

A key feature of the approach is its simplicity: it requires only a straightforward actor-critic architecture and learning algorithm, making it easy to implement and scale to more difficult problems and larger networks.

IV. Results

In order to evaluate the method, the authors constructed a variety of challenging physical control problems that involve complex multi-joint movements, unstable and rich contact dynamics, and gait behavior.

Among these are classic problems such as the cartpole swing-up problem, as well as many new domains. A long-standing challenge of robotic control is to learn an action policy directly from raw sensory input such as video. Accordingly, we place a fixed viewpoint camera in the simulator and attempted all tasks using both low-dimensional observations (e.g. joint angles) and directly from pixels.

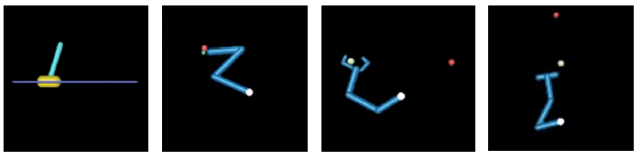

Fig. 4. Tasks from left to right: the cartpole swing-up task, a reaching task, a gasp and move task, a puck-hitting task

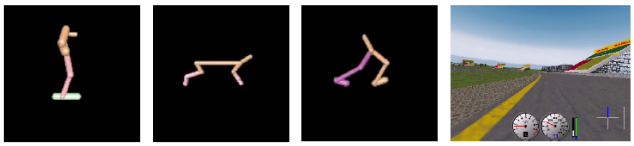

Fig. 5. Tasks from left to right: a monoped balancing task, two locomotion tasks and Torcs (driving simulator)

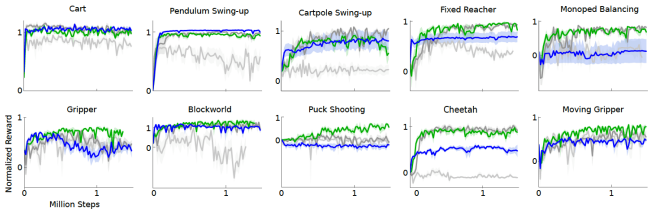

Fig. 6. Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

V. How it relates to class

DDPG is a model-free algorithm that can learn an optimal policy from the "environment, action and reward" without modelling the environment.

We learned bellman's equation in class. The critic $Q(s, a)$ is learned using the Bellman's equation as in Q-learning. Q-learning applies a value-iteration method that finds the optimal policy. We also covered DQN and policy gradient in class. DQN tackles problems with discrete actions. DPG is combined with DQN to handle the task for continuous action domain. While DQN uses epsilon-greedy to explore, DDPG uses Gaussian noise to explore.

VI. Pros and Cons

Pros [3]

(i) Can work very well for high dimensional continuous action problems;

(ii) Learns an action-valued function instead of a state-valued function.

Cons

(i) Does not explicitly provide a formalism for the exploration to be used;

(ii) Needs a target network to increase stability while learning an action-valued function;

(iii) The policy is only as good as the Q-function is accurate.

Moreover, the paper "Y. Hou, et al., A novel ddpg method with prioritized experience replay" [4] proposes a prioritized experience replay method to improve the efficiency of the experience replay mechanism in DDPG. The proposed DDPG with prioritized experience replay is tested with an inverted pendulum task via Openai Gym. The experimental results show that DDPG with prioritized experience replay can reduce the training time and improve the stability of the training process, and is less sensitive to the changes of some hyperparameters such as the size of replay buffer, minibatch and the updating rate of the target network.

The author also points out in the DDPG paper that DDPG requires a large number of training episodes to find solutions. However, we believe that a robust model-free approach may be an important component of larger systems which may attack this limitation.

References

[1] T. Lillicrap et al., Continuous control with deep reinforcement learning, 2015.

[2] V. Mnih et al., Human-level control through deep reinforcement learning, Nature, 2015

[3] Glen Berseth Blog, Demystifying the Many Deep Reinforcement Learning Algorithms

[4] Y. Hou, et al., A novel ddpg method with prioritized experience replay