MF20150100-邱祥智-DDPG

What is DDPG ?

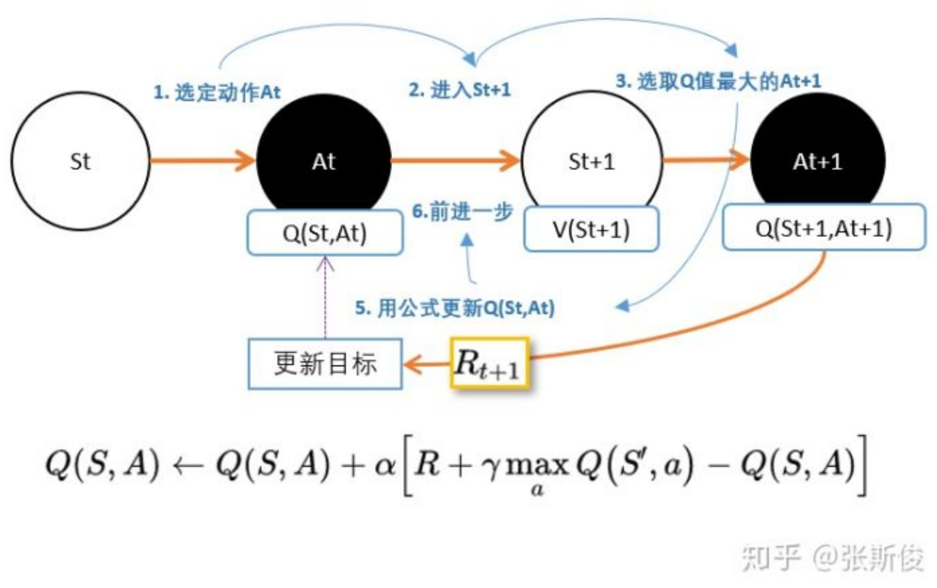
DDPG, its full name is 'Deep Deterministic Policy Gradient'. We have learned the 'Policy Gradient' before, and 'Deep' means to use the deep neural network to fit the Q function.

$\theta = \theta + \alpha \nabla_\theta J_\theta$ (1)

$\nabla_\theta J_\theta = \frac{1}{N}\sum_{i=1}^{N}(\sum_{t=1}^{T}\nabla_\theta log\pi_\theta(a_t^i|s_t^i)(\sum_{t'=t}^{T}\gamma^{t'-1}Q(s_{t'},a_{t'})))$ (2)

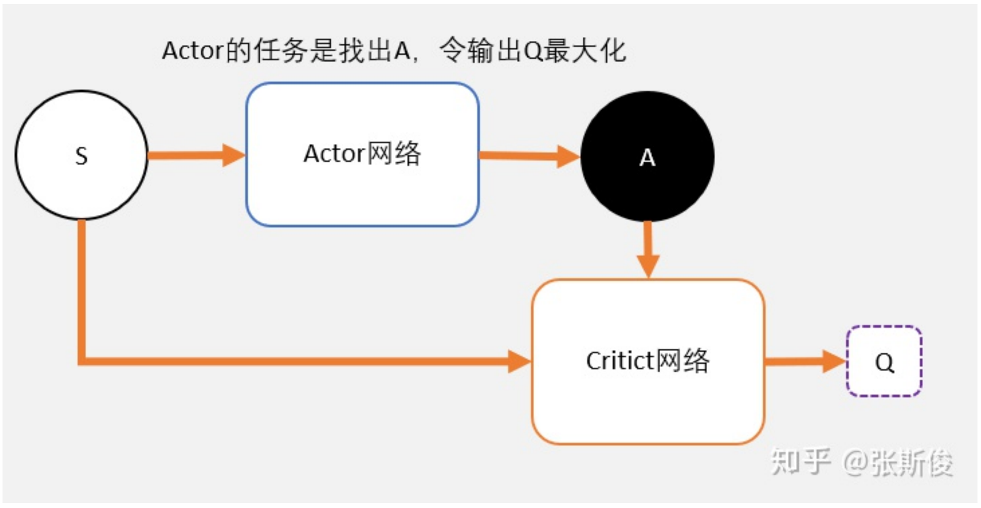
What is the 'Deterministic'? We use the outputs of network to fit the mean and variance of normal distribution. And we sample actions from this distribution, it's not deterministic. Now, we just view the outputs as the actions, it is deterministic.

Let's review the DQN, the process shows it can not handle the continuous control problem. That's for we should get the maximize Q function with the optimizing action a_i to arrive sate $s_{i'}$.



So, how we solve this problem? Actually it's easy to get experience from DQN. We use the deep neural network to fit the Q function.

But the goal of the DDPG is to increase the grad. Because of the network actor is finding the maximized value. So it's a off-line strategy without the importance sampling.



Critic Network:

- We use the critic network to evaluate the Q function in current state and action.
- Input elements: action, state.
- Update method:
- $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j,a_j) - y_j\right)$

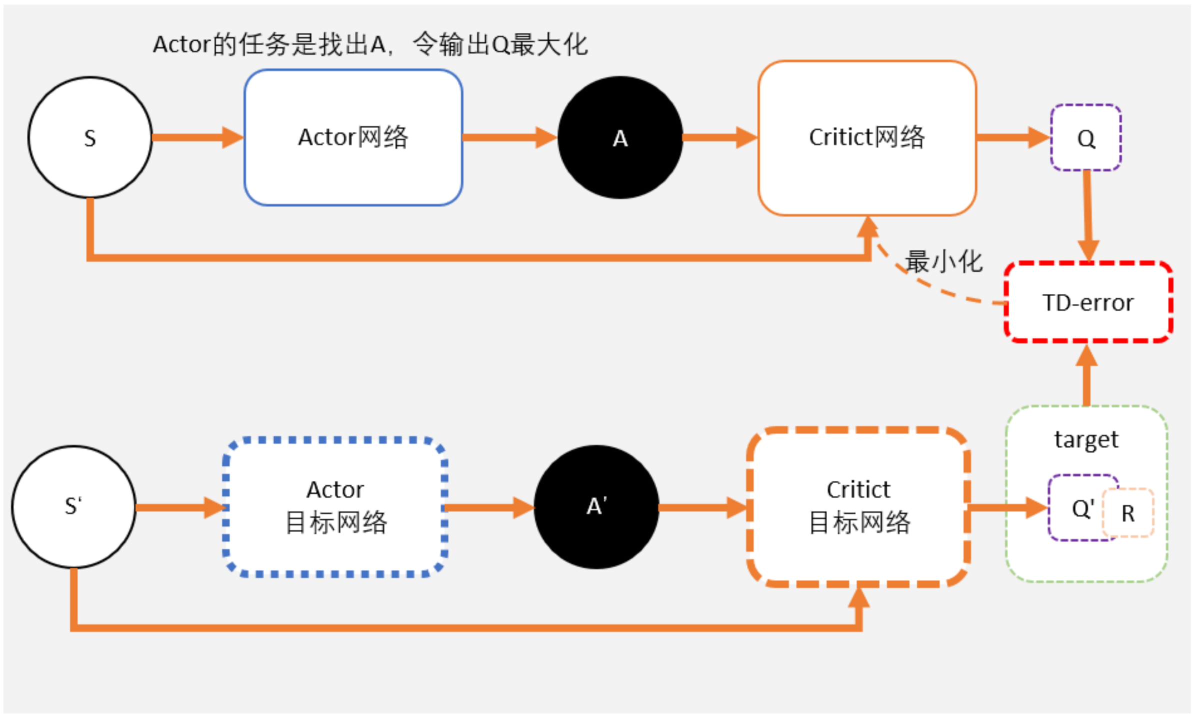
Actor Network:

- We use the actor network to get the action.
- Input elements: current state.
- Update method:
- $\theta \leftarrow \theta + \beta \sum_j \frac{\mathrm{d}Q_\phi}{\mathrm{d}\mu_\theta(s_j)} \frac{\mathrm{d}\mu_\theta(s_j)}{\mathrm{d}\theta}$

The logistical code step as follow:

> - Loop:
>   1. take some action a_i and observe (s_i, a_i, s'_i, r_i), add it to $\mathcal{B}$
>   2. sample mini-batch $\{(s_j, a_j, r_j, s'_j)\}$ from $\mathcal{B}$ uniformly
>   3. compute $y_j = r_j + \gamma Q_{\phi'}(s'_j, \mu_{\theta'}(s'_j))$ by *target* networks $Q_{\phi'}$ and $\mu_{\theta'}$
>   4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j,a_j) - y_j\right)$
>   5. $\theta \leftarrow \theta + \beta \sum_j \frac{\mathrm{d}Q_\phi}{\mathrm{d}\mu_\theta(s_j)} \frac{\mathrm{d}\mu_\theta(s_j)}{\mathrm{d}\theta}$
>   6. update ϕ', θ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$, $\theta' \leftarrow \tau\theta' + (1-\tau)\theta$

And the the process as below (maybe not use TD-error):



Where the difference between DDPG and PG or AC?

Compare with PG:

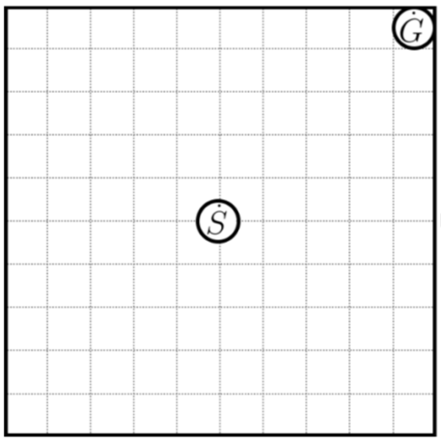
1. PG uses the actor network to fit the distribution of action, is not deterministic. While ‘DDPG’ use the actor network to get the deterministic action which can maximize the Q function.
2. PG uses the accumulative rewards to evaluate the advantage of next state compared with current state. While ‘DDPG’ use the another network which called critic to evaluate the action.
3. PG uses the same neural network to learn and make policy decision (on-policy). While ‘DDPG’ use the actor-critic to learn and target actor-critic network to make policy decision (off-policy) .
4. Of course, its update methods of parameters are different, refer to the forward theory classify.
5. PG sample a batch size sequential info set from the actor network, while ‘DDPG’ get from rely buffer uniform randomly.

Compare with AC:

1. Update method of AC, which is different:
   $$\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$$
2. AC use same net work to learn, critic and fit. It is an on-policy algorithm while ‘DDPG’ is not.
   1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
   2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
   3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
   4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
   5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
3. AC get the sequential samples from the network, while DDPG get uniform randomly from rely buffer.

How about the code of DDPG?

Problem 1: Maze Navigation, we need to train the robot to get the right route from start to goal.



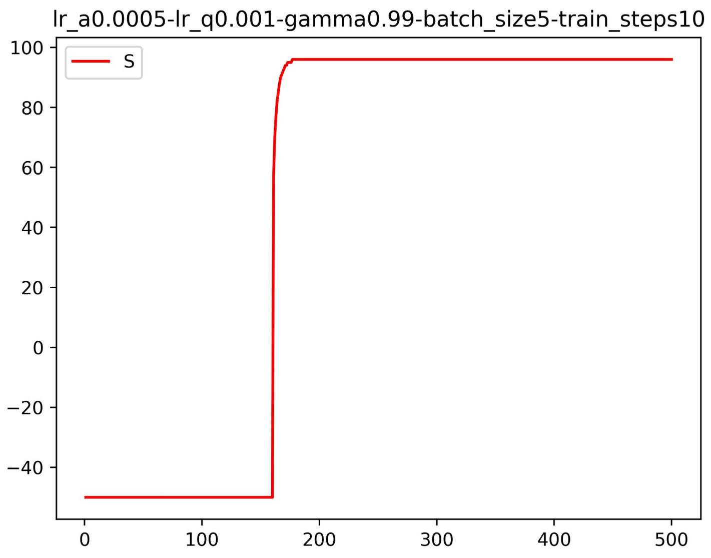
So, there are some parameters be definite in the below:

```
#Hyperparameters
lr_mu         = 0.0005
lr_q          = 0.001
gamma         = 0.99
train_times   = 10000
each_train_times = 10
batch_size    = 50
buffer_limit = 5000
tau           = 0.005 # for target network soft update
```

The environment settings as below:

```
# Maze Nvigation
action_dims          = 2
state_dims           = 2
out_of_border_reward= -100
goal_area            = [0.4,0.5]^2
goal_area_reward     = 100
map_area             = [-0.5,0.5]^2
start_point          = [0,0]
other_area_in_map    = -1
action_lock          = [-0.1,0.1]^2
```

The result of ‘DDPG’ in Maze Navigation (it has trained 10000 epochs actually, and record the average reward every 20 steps):



It seems great, and it's really smoothly. I feel awesome with my code.

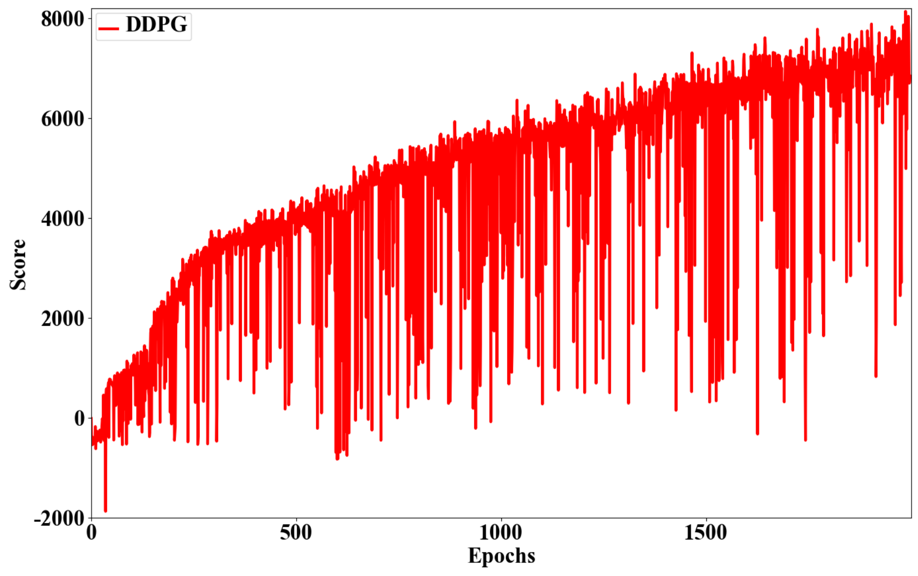
Really strange thing is I always get reward number: -104. Which means the robot run 4 steps in map area and steps out at the fifth step. Another strange thing is the robot thinks this is the best result in the mechanism of ‘DDPG’. How's the world? Maybe the inner goal area will be more friendly for the robot.

Problem 2: We need to make the robot walk forward.

I don't know the full informations about this environment. But I know the max location is infinite and the min is negative infinite. The max action step is 1, the min action step value is -1. And the observation of environment are 17 axis which action are 6. And I find a really funny video online, which used ‘PPO’ to train the robot take forward by the environment ‘HalfCheetah-v2’. And he used more than 0.2 million steps to train it. At that before, the robot even seemly don't take actions.

Here's the link: PPO in HalfCheetah-v2 training.

The result of 'DDPG' in 'HalfCheetah-v2' which coded by myself:



Awesome, it's a really great result for me. I see the robot runs the first.