Twin-Delayed Deep Deterministic Policy Gradient (TD3)

Mahroo Bahreinian
mahroobh@bu.edu

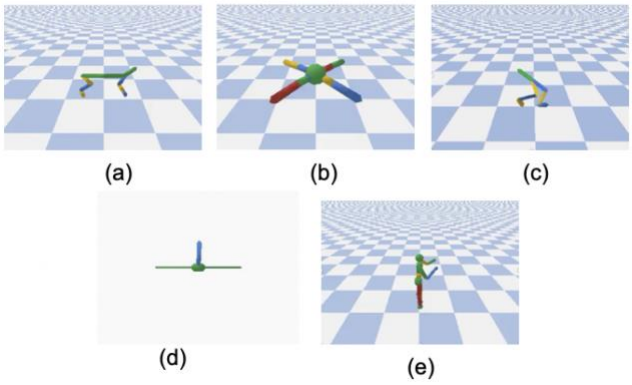

Figure 1. train the agent in Pybullet environment. (a) half-cheetah, (b)ant, (c)walker2D, (d)inverted-Pendulum and (e) humanoid.

1. Task

In Reinforcement Learning (RL), the agent learns how to interact with the environment while achieving the highest reward. The interaction of the agent with the environment Is defined through the sequence of the actions that agent can fallow to learn a task on the environment. The main concern in RL is to find the policy to produce the sequence of actions to achieve the highest reward. The RL algorithms can be divided to different categories. In terms of model, the RL can be dived to tow model based and model free class. Model-free methods take the data from the environment and in model-based algorithms the agent learns from the imagination of the environment, which means in these algorithms, agent predict the next state and not taking it directly from the environment. RL can be categorized to value-based (e.g., deep Q-learning) and policy-based (e.g., policy gradient) algorithms, which in value based the parameters of the policy function are updated in terms of value function but in policy-based algorithms the parameters are updated directly. The final class is off-policy and on-policy methods, which in off-policy algorithms the agent learns from the past experiments which are stored in the memory however in on-policy, the agent learns through the new observation of the environment.

Many models are introduced to find the policy function for the continuous action space however these algorithms suffer from the errors which are induced by the function approximation to the result. In this project I re-implement the “Addressing Function Approximation Error in Actor-Critic Methods” paper which introduced the algorithm to decrease the error from the function approximation. In this work, I make a thorough study on policy gradient and Q-learning techniques and re-implement the presented algorithm in the paper in Pytorch and run my code over half-cheetah, ant, walker, humanoid and inverted-pendulum environments in PybulletGym.

2. Related Work

Many algorithms have been introduced to learn the policy includes Q-learning [1], SARSA [2] and policy gradient [3].These algorithms use linear function approximation methods to estimate the action value, although the convergence of these algorithms is guaranteed, for the complex real world they are inefficient. So, to tackle this, the non-linear function approximation methods introduced. The deep Q-learning (DQN) [4] algorithm improves the flexibility of the agent; however, it causes the problem of overestimation bias and high variance due to the accumulation of error in update phases. This problem in well studied in discrete action space but not enough in continuous space. Some algorithms are introduced to overcome this problem, double-DQN [5] uses two independents estimators to make unbiased value estimates however it does not work properly with actor-critic models as in double-DQN, the policy is updated with separate target network so the actions are evaluated without the effect of overestimation, however, in actor-critic model, the policy updates slowly so the target and policy remain so much similar and overestimation is unavoidable.

To overcome this problem, double-DQN in actor critic format uses two independently critics, however even the unbiased estimator with high variance can cause the overestimation in future updates. The proposed algorithm in this work proposed the clipped version for double DQN where it uses the minimum value function between the critics to update the parameters. This tends to the underestimation, however, the policy avoids the actions with less rewards, so it does not propagate during the training process and cannot cause a problem in finding the optimal policy.

3. Approach

TD3-TwinDelayed-DDPG is includes two parts, deep Q-learning and policy gradient. In deep Q-learning to find the optimal policy for continuous action space.

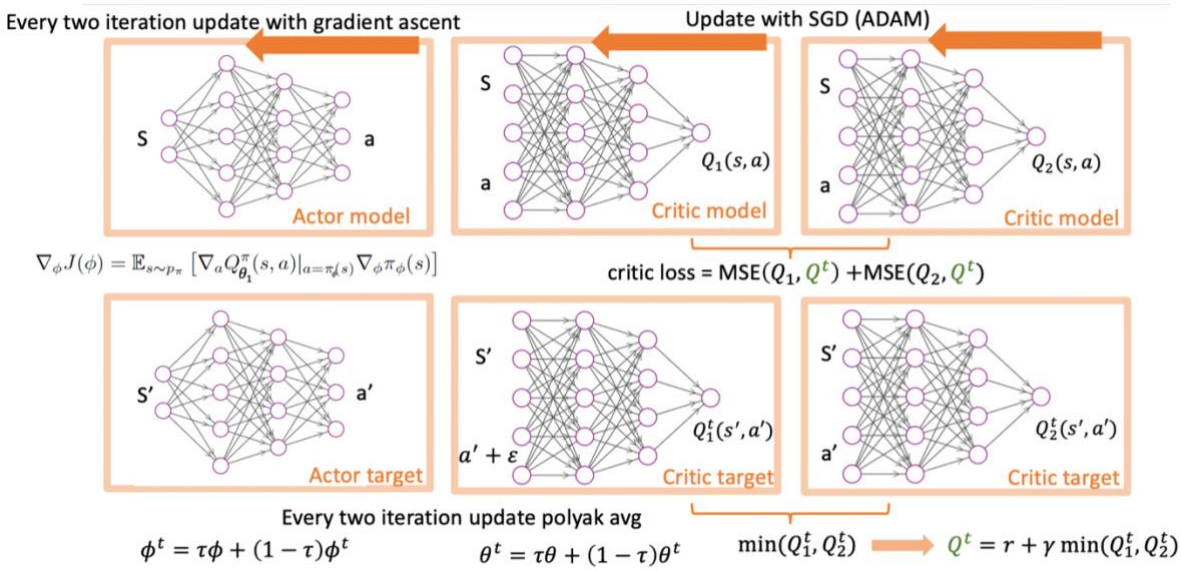

Figure 2. The view of Twin-Delayed DDPG (TD3) network

3.1. Deep Q-learning Network (DQN)

Deep Q-learning is a model-free, value-based, off-policy algorithm which is applicable to environment with discrete action space. Deep Q-learning learns the value function which gives the highest reward through a neural network. The neural network learns the parameters of the Q-value function via decreasing the loss between the target value and predicted Q-value via SGD where the target is the optimal reward in each time step. As the target varies in each time steps, we can use one separate network for the target to compute the optimal reward in each time step and copy the parameters of the target network from first network after improving the Q-value parameters in first network for some number of iterations. Deep Q-learning is an off-policy method which needs a set of experience consists of current state, current action, next action and reward. So, we need to store data of the past experience in a memory before starting the training process.

3.2. Actor-Critic

In policy gradient algorithms, the optimal policy is found explicitly through maximizing the reward. Actor-critic [6] algorithm is an off-policy model-free algorithm consists of two essential parts, value function and policy model. The value function part updates the parameters of the value function through a neural network and the policy part updates the policy parameters in the direction of increasing the value function.

3.3. Deterministic policy gradient (DDPG)

DQN algorithms learn the optimal policy implicitly in discrete action space, however the environment could be more complex and needs to have the continuous action space, to deal with this, the deterministic policy gradient is proposed. DDPG [7] combines the actor-critic algorithm and DQN for continuous action space while learning the deterministic policy. In order to to a better exploration, DDPG adds a noise to the deterministic policy. In DDPG we have for networks, actor network, actor target network, critic network and critic target network. The target networks are updated in each iteration via polyak averaging.

Algorithm 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

3.4. Twin-Delayed DDPG (TD3)

Although the DDPG algorithm has a good performance, it suffers from the overestimation error from learning the Q-value function. To tackle this problem, TD3 [8] model is emerged and adds three essential parts to the DDPG model to reduce the overestimation bias and high variance. First instead of learning one Q-function

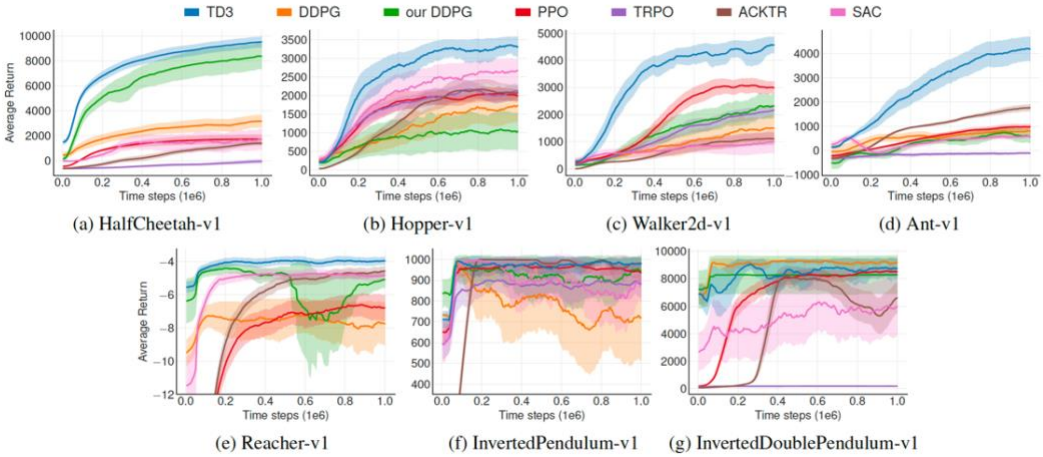

Figure 3. numerical comparison results

through neural networks, it uses two separate neural networks and learn two Q-functions and pick the smaller one for computing the loss function, this trick is called clipped double Q-learning. Second, the update of the policy parameters happens after two updates of the Q-function which is called delayed trick and third, it adds noise to the actions for target network to reduce the high variance which is called target policy smoothing. These three tricks make the TD3 algorithm to perform well in continuous action space. The detail of the algorithm is shown in blow and also in Fig.2.

3.5 Implementation of TD3

To implement the TD3 model, we need 6 neural networks, actor model, actor target, two critic model and two critic targets. Model and target networks are initialized the same in the beginning and updates during the training process with the TD3 algorithm. The structure of the all networks are similar. For actor networks we have two hidden layers, first with 400 neurons and second with 300 neurons, the activation function for first and second layer is RELU and the activation function for the last layer is tanh. For the critic networks all the structures are similar expect the output does not have an activation function. As the TD3 algorithm is off-policy, we need to store the past experience in the memory and train the networks on that data set. The details of the algorithm are shown in Fig.2 and Algorithm TD3.

In openAI spinning up, there is function to call to use the TD3 model, however in this work I implemented it from the scratch in Pytorch and run it in different environments. The details of the code are available on the GitHub. To implement the TD3 algorithm I use some libraries in Python as: NumPy, pybullet_envs, gym and torch.

Algorithm TD3

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

4. Dataset

For this project I use the Pybullet open-source environment which contains MuJoCo environments for deep reinforcement learning purposes.

5. Results

First, I describe the result of the paper and then discuss about my own code results. In the paper, their run the TD3 algorithm over seven different environments nclude Half-cheetah, Hopper, Walker, ant, Reacher, Inverted Pendulum and Double Inverted Pendulum and compare the result with six existing algorithms such as DDPG and soft-actor-critic. The results show that the TD3 model surpasses all the algorithms and performs with higher rewards in the environment. The numerical results are shown in Fig.3 and table 1. The TD3 model obviously

Table 1. Max Average Return over 10 trials of 1 million time steps. Maximum value for each task is bolded. ± corresponds to a single standard deviation over trials.

Environment	TD3	DDPG	Our DDPG	PPO	TRPO	ACKTR	SAC
HalfCheetah	**9636.95 ± 859.065**	3305.60	8577.29	1795.43	-15.57	1450.46	2347.19
Hopper	**3564.07 ± 114.74**	2020.46	1860.02	2164.70	2471.30	2428.39	2996.66
Walker2d	**4682.82 ± 539.64**	1843.85	3098.11	3317.69	2321.47	1216.70	1283.67
Ant	**4372.44 ± 1000.33**	1005.30	888.77	1083.20	-75.85	1821.94	655.35
Reacher	**-3.60 ± 0.56**	-6.51	**-4.01**	-6.18	-111.43	-4.26	-4.44
InvPendulum	**1000.00 ± 0.00**	**1000.00**	**1000.00**	**1000.00**	985.40	**1000.00**	**1000.00**
InvDoublePendulum	**9337.47 ± 14.96**	**9355.52**	8369.95	8977.94	205.85	9081.92	8487.15

performs better than other algorithms. I run the algorithms in five environments and the rewards are reported below, I train the model for 500,000 iterations, the video of the results is available at GitHub.

environment	Highest reward
hopper	2340.856943
Walker2D	1873.562363
half Cheetah	2269.651263
inverted Pendulum	875.3965128
Ant3D	2363.117268

The recorder rewards are similar to the 500,000 iteration on Fig 3.

6. Detailed Roles

I did this project alone, but to mention I am part of the dream to control project as well.

7. Code repository

Here is the link to the GitHub repository: https://github.com/Mahrooo/Implementing_TD3_deepRL.git

References

[1] Watkins, JCH and Peter, "Q-Learning," *Machine Learning,* 1992.

[2] H. Van Seijen, H. Van Hasselt and S. Whiteson, "A theoretical and empirical analysis of expected sarsa. In Adaptive Dynamic Programming and Reinforcement Learning," *ADPRL'09. IEEE Symposium on, pp. 177–184.,* 2009.

[3] Sutton, Richard S, McAllester, David A, Singh, Satinder P, Mansour and Yishay, "Policy gradient methods for reinforcement learning with function approximation," *NIPS,* 1999.

[4] V. K. K. S. D. R. A. A. V. J. B. M. G. G. Mnih, "Human-level control through deep reinforcement learning," *Nature, 518(7540):529 533,* 2015.

[5] H. G. A. a. S. D. Van Hasselt, "Deep reinforcement learning with double Q-learning," *arXiv preprint arXiv,* 2015.

[6] V. R. a. T. J. N. Konda, "Actor-critic algorithms," *Advances in neural information processing systems,* 2000.

[7] J. J. H. ,. A. P. N. H. T. E. Y. T. D. S. D. W. Timothy P. Lillicrap∗, "Continuous control with deep reinforcement learning," *arXiv,* 2019.

[8] S. Fujimoto , H. v. Hoof and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *arXiv,* 2018.