Tsinghua University

Deep Reinforcement Learning

Spring 2024

Assignment 3

Subject: Deep Q-Network

Author: Tonghe Zhang(张同和)

Email: zhang-th21@mails.tsinghua.edu.cn

Affiliation: Department of Electronic Engineering, Tsinghua University

May 8, 2024

Contents

1 Deep Deterministic Policy Gradient (DDPG) 1

2 Twin-delayed DDPG (TD3) 2

3 Soft Actor Critic (SAC) 2

1 Deep Deterministic Policy Gradient (DDPG)

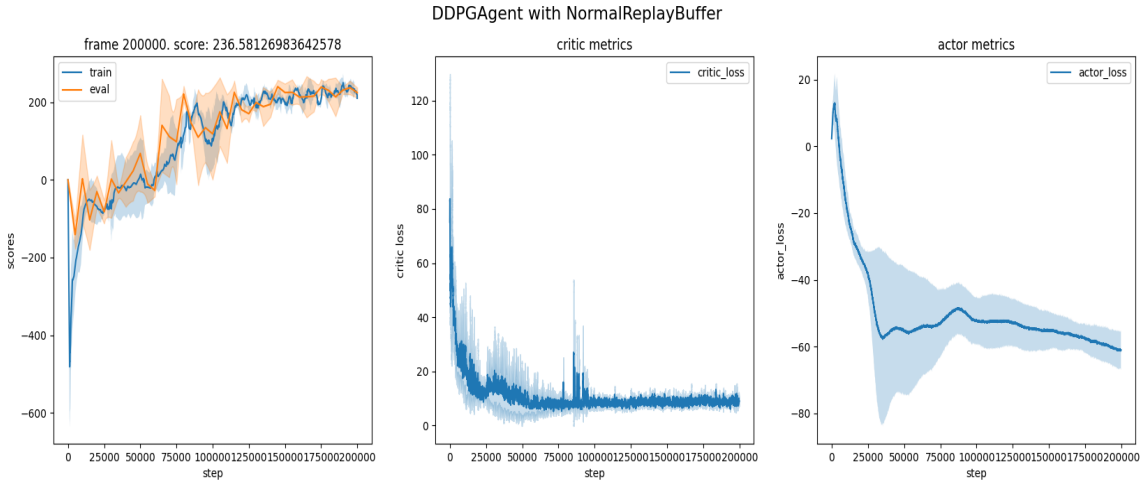

Fig 1: TD3 Result

DDPG training curves looks very normal. In the final video the lunar lander also lands safely.

2 Twin-delayed DDPG (TD3)

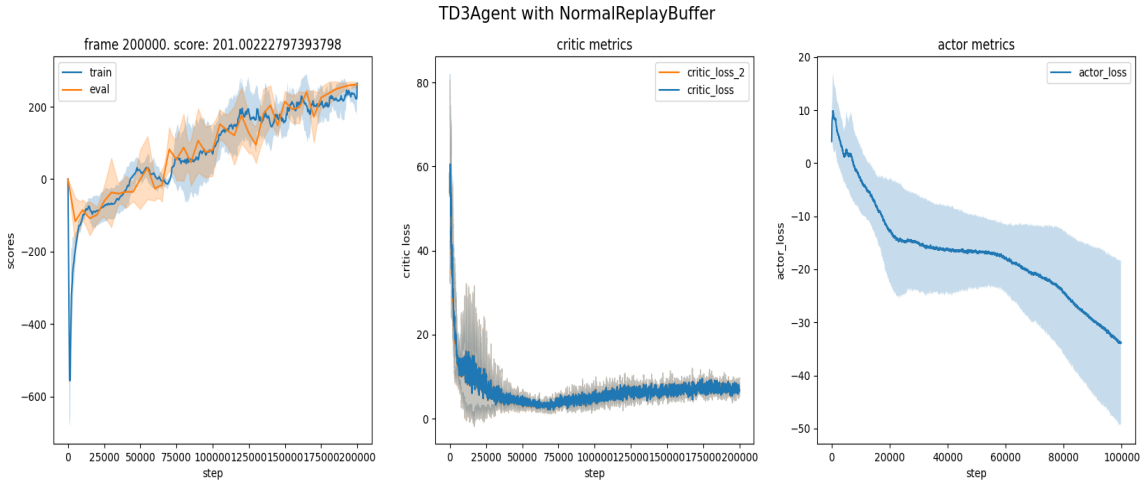

Fig 2: TD3 Training Result

In TD3 training we should remember to add ".detach()" during the calculation of the critic loss and actor loss, as these losses share the same actor or critic net, which could cause repeated gradient computation error.

3 Soft Actor Critic (SAC)

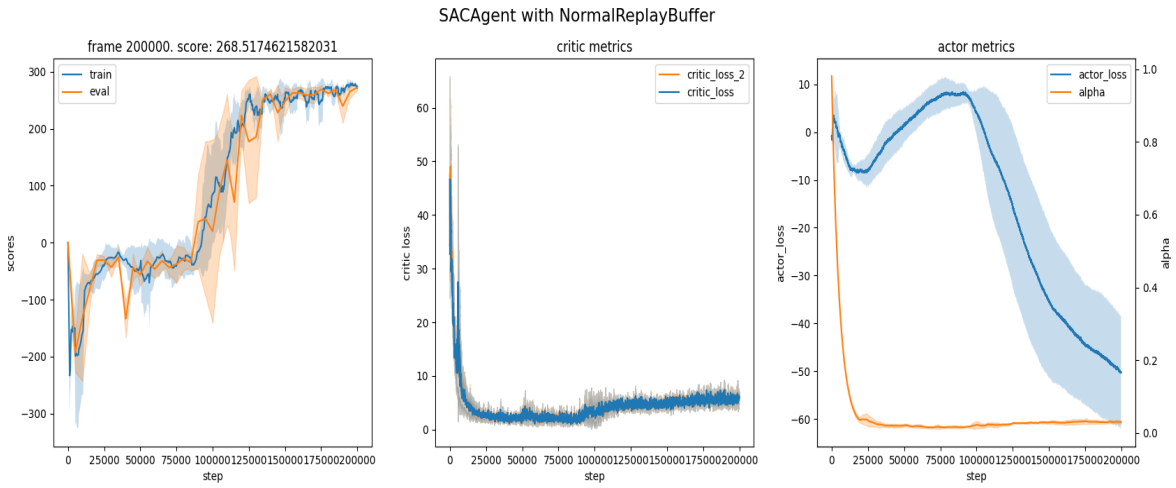

Fig 3: Soft Actor Critic Result

Remark 3.1 The actor loss increased a while during steps [25000,100000]. This is because the critic loss decreases too quickly, causing the value net to overfit while the policy is still largely suboptimal. During this phase the average returns get stuck in local minima. However, since the SAC agent sample actions from a stochastic process (gaussian policy), the exploration of the agent successfully helps the optimizer to escape the local minima and finally reaching satisfying status at the end of the trainin.

After training the SAC and TD3 agent I have severl reflections on the concepts and methods of continuous-control Deep RL.

- How to do continuous control using stochastic policy in RL: In TD3, we use a neural net to receive continuos-value inputs and output deterministic action. Then we add white gaussian noise to create stochasticity for exploration. We clip the signal+noise as the selected action. In SAC, we sample action from a gaussian distribution, whose mean and variance are parameterized neural net's outputs. We train the agent to dynamically adjust the model's parameter, so that the policy is also improved. For robustness we also apply a tanh() transform to rescale the unbounded gaussian values. In either way, we will never store actions (which restricts the policy to be tabular). We either use neural nets as continuous functional to input and output continuos values, or we parameterize a continuous yet simple distribution class (tanh gaussian, which has closed-forms) to train the distribution class's parameters instead(reparameterization trick).
- It is observed in practice that the actor loss and critic loss usually cannot be simultaneously optimized in the same direction, it often happens that the critic loss goes down quickly while the actor loss pumps up in the other direction. This is because the critic loss only forces our value function approximator to learn the correct functional that obeys the Bellman equations, but it has nothing to do with finding the optimal policy funcitonal; The actor loss, though aiming to maximize the rewards by optimizing the policy, does not necessarily guarantee that this reward is generated under a correct value function class. The discrepancy in the optimization objective implies that logically we can barely see that the two losses goes down neck and neck. The oposite direction of the two optimizers could incur numerical instability, and that is why we introduced delayed updates in TD3.
- In SAC, though we introduce parameterized gaussian distribution to generate raw actions, that gaussian distribution is inherently unbounded could bring numerical stability to the training process. For this reason we apply another $tanh(\cdot) : \mathbb{R} \to [-1, 1]$ transform to the policy's output to bound the action. This distributino transform brings a jacobian determinant coefficient to the calculation of the log probability of the normalized policy, which will be then used as an unbiased estimator of the negative entropy bonus.
- In later version of SAC we can dynamically adjust (i.e. to learn) the magnitude of entropy regularization term. So we introduced another alpha loss and viewed alpha as requires_grad=True object.