Deep RL Baseline Reimplementation

Shengjian Chen
University of Washington
Seattle, WA 98105
`sjchen19@uw.edu`

Abstract

This is the final project for CSE 573 in Winter 2020. In this project, I reimplement 5 state-of-the-art deep reinforcement learning algorithms (A2C, DDPG, PPO, TD3, SAC) and experiment on 3 tasks with continuous action space on OpenAi Gym (Pendulum-v2, HalfCheetah-v2 and Hopper-v2). Furthermore, I investigate and compare different factors' effect on the performance of the algorithms.

1 Introduction

We have talked a lot about Markov Decision Process (MPD) and reinforcement learning in class. However, those methods are very hard to scale to complicated tasks. With the rise of deep learning in recent years, deep reinforcement learning is proved to be feasible for that. DQN[1] is the first practical algorithm that plays Atari game with DRL. And Later researchers proposes lots of techniques and apply them to a wide range of tasks like classical control, robotics and video games.

However, deep reinforcement learning is not always working as they promise. There has been lots of complains about the issue of reproducibility of DRL algorithms in the community. As addressed in [7], there are many factors that have great impact on the performance in the implementation of DRL. Carefully investigating their effects is very helpful to the healthy development of this field.

In this project, I will first implement 5 state-of-the-art DRL algorithms as the baseline. During experiments, I will focus on continuous action tasks that can be extended to control and robotics problems. Through these experiments, I want to study several critical aspects that effect the performance of the algorithms. This project is based on the idea of [7] but adds in two new algorithms and try to extend the discussion with my own results. The code is available on https://github.com/tony23545/Deep-Reinforcement-Learning-Reimplementation.

2 Preliminaries

In this section, I will briefly summarize the techniques used in the set of algorithms I choose and some critical factors during implementation.

2.1 Deep Reinforcement Learning

I have chosen totally 5 algorithms: A2C, DDGP, PPO, TD3 and SAC. They altogether cover almost all practical techniques used in modern DRL. I implement all these algorithms in the actor-critic style.

Actor critic is a kind of on-policy algorithms that requires collecting experiment trajectories during training. [2] proposes to use parallel workers to generate training data and update the policy asynchronously or synchronously. A2C is the synchronous variant and is proved to be more efficient. Actually, parallel workers can be used in any algorithms as a module to collect data.

Most previous algorithms use stochastic policy. DDPG[3] uses deterministic policy and gains very impressive performance on continuous action tasks. Getting rid of the expectation on actions, it is a Q-learning style off-policy algorithm with good sample efficiency.

PPO[4] is among another family of algorithms called policy optimization. TRPO is also very famous but PPO is easier to implement and has similar results in most situations. Although it is an on-policy methods, it reuses the data for multiple times within the trust region, thus can be more efficient than vanilla actor critic.

TD3[5] address some problems in DDPG with 3 tricks: clipped double Q-learning, delayed policy update and target policy smoothing. These techniques makes the training process more stable.

SAC[6] is proposed nearly concurrently with TD3 and they share some similarities. But SAC learns a stochastic policy and its most important contribution is introducing the entropy regularization term to Q value to encourage exploration.

2.2 Things that Matters

As reported in [7] and many other works, performance of DRL is far from stable and is affected by many factors. For example, different algorithms have their own hyper parameters needed to be fine-tuned. Different environments can cause dramatic different in the same algorithm. Experiment random seeds decide what scenarios the model see and could lead to different results. There are so many choices in these issues so that it is very hard to reproduce the results in many published works. Actually, we are still far from a general model that applicable to all situations.

3 Experiment Setup

All my experiments are carried out on 3 environments from OpenAi Gym and Mujoco. *Pendulum-v2* is a relatively easy task and I use it for early debugging. *HalfCheetah-v2* and *Hopper-v2* are two more complicated tasks with higher dimension of observation space and action space. They also have their own interesting characteristics which I will discuss in detail later. To investigate the effect of one factor, I will try to only change that factor within a group of experiments but keep the other settings the same to default values. As revealed in many works, A2C converges slower than the other 4 algorithms in most cases. Therefore, I won't report results on A2C in this report. I train all the algorithms with different settings on 2 machines with only CPUs.

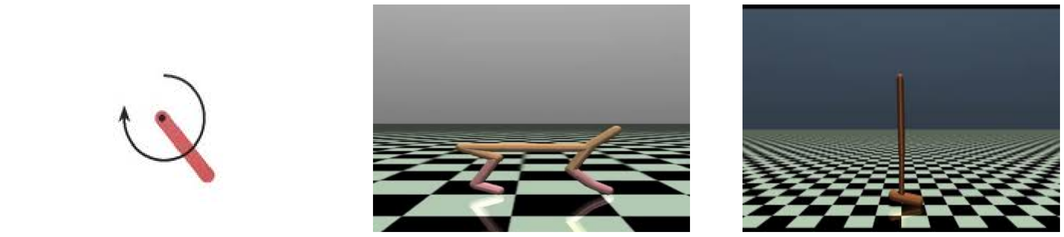


Figure 1: 3 Experiment Environments

4 Results and Discussion

In this section, I will discuss 4 critical factors in DRL respectively with my experiment results. Note that I continue my experiments after submitting the presentation, so the results will be slightly different.

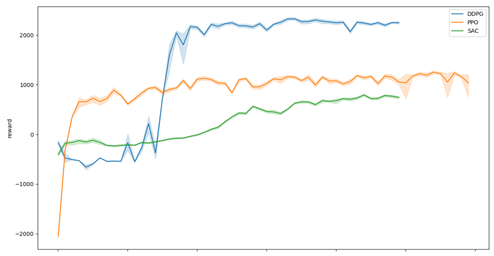
4.1 Environments

HalfCheetah-v2
This environment has 11 Dof observation space and 6 Dof action space. The dynamic is stable so the cheetah won't fall to terminate the episode. However, this environment could have multiple local optima and models can be easily get tracked. In this task, I find models with deterministic policy like

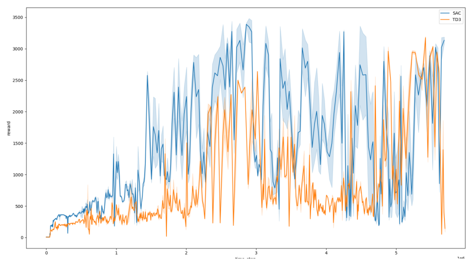
DDPG and TD3 perform better in terms of the final rewards in many cases given the same random seed maybe due to more exploration.

Hopper-v2
This environment has 11 Dof observation space and 3 Dof action space. In contrast, the dynamic here is unstable. The hopper can fall easily and terminate the episode given bad policy. Therefore, the model can't get any useful information at the early stage of learning. In this situation, I find all learning curves oscillating a lot although in many publications they are quite smooth. According to my observation, models with stochastic policy work better. This time, we may need to exploit more to let the hopper get further.

We can see that performance of a algorithms also depends on the environment. Selecting the proper algorithm for given task is a very important step.



(a) HalfCheetah-v2 result



(b) Hopper-v2 result

Figure 2: Results of different algorithms on the 2 environments

4.2 Hyperparameters

There are lots of hyperparameters in DRL. For many of them, there are recommended value based on lots of researchers' experiments. A comprehensive analysis will involve a great amount of experiments. Due to the time limit, I can only briefly analyse some important hyperparameters.

During experiment, I find that a smaller learning rate like 0.0001 is good for PPO but the default value 0.001 usually break down the model. For off-policy algorithms, I also explore the capacity of replay buffer (Figure 3) on the *HalfCheetah-v2* environment but it seems to make no difference even with a relatively small capacity like 50000.

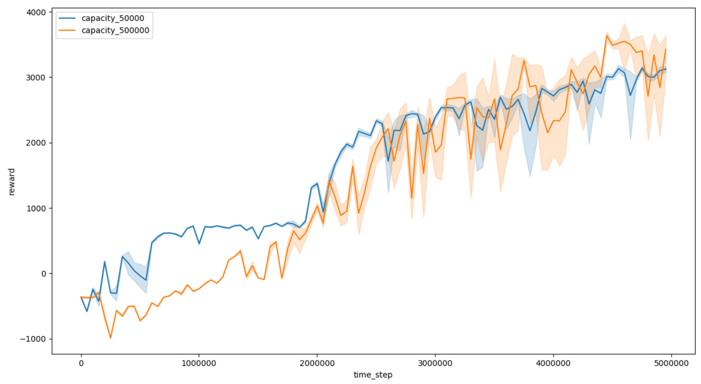


Figure 3: HalfCheetah-v2(TD3, different replay buffer capacity)

4.3 Network Structure

As reported in [7], using a simple network in PPO like (64, 64) [1] is good enough for tasks like *HalfCheetah*. More complex structures like (100, 50, 25) just make the thing worse. I observe similar

[1] 1 hidden layer with 64 inputs and 64 outputs

result (Figure 4) on TD3. I pick a very good random seed that performs very well on (64, 64) but it breaks down on complex structure like (512, 512, 512). Actually, it is very hard to decide what is the proper complexity of the networks given a new environment.

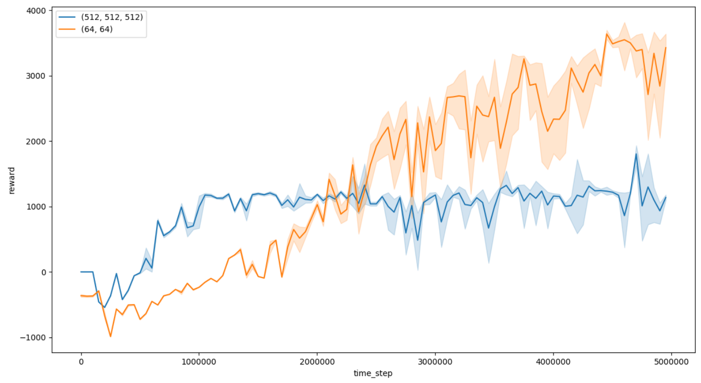


Figure 4: HalfCheetah-v2(TD3, different network structures)

4.4 Random Seed

DRL relies heavily on the experiment trajectories generated randomly. Random seed plays a critical role here. My result (Figure 5) [2] also reveals that different random seeds can lead models to different local optima. And it seems very hard to jump out and find better solution since many models tends to reduce exploration in later stage of the training process. What's worse, in the *HalfCheetah-v2* environment, I find it very easy to fall into the bottom-up local optima for most of random seeds. This rise a great challenge to generalize our learning models.

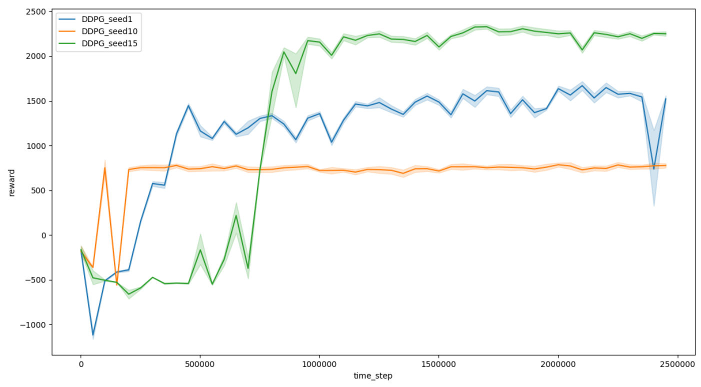


Figure 5: HalfCheetah-v2(DDPG, random seeds)

5 Conclusion

In this project, I make a brief review of the development of DRL and get my hand dirty by implementing 5 state-of-the-art algorithms. By a serial of experiments and analysis, I gain the preliminary feeling how different aspects affect the performance of DRL models. This project has great enhanced and extended my understanding of the course materials.

References

[1] Mnih, Volodymyr, et al. “Playing Atari with Deep Reinforcement Learning.” 2013.

[2] curves are smooth here probably because I use a smaller replay buffer in this group of experiments

[2] Mnih, Volodymyr, et al. “Asynchronous Methods for Deep Reinforcement Learning.” ICML 2016.

[3] Lillicrap, Timothy P., et al. “Continuous Control with Deep Reinforcement Learning.” 2015.

[4] Schulman, John, et al. “Proximal Policy Optimization Algorithms.” 2017.

[5] Fujimoto, Scott, et al. “Addressing Function Approximation Error in Actor-Critic Methods.” 2018.

[6] Haarnoja, Tuomas, et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.” 2018.

[7] Henderson, Peter, et al. “Deep Reinforcement Learning That Matters.” 2017.

[8] Duan, Yan, et al. “Benchmarking Deep Reinforcement Learning for Continuous Control.” 2016.