Deep Reinforcement Learning Algorithm Comparison

Han Jiang [1]

Abstract

Deep reinforcement learning (DRL) is currently the most popular research direction in computer science and can be applied in many domains. This paper compares the performance of four DRL algorithms in continuous and discrete action spaces in the PyBullet environment, and then summarizes their advantages and disadvantages.

1. Related Work

1.1. DQN

Off-policy is a feature of Q-Learning, which is also used in DQN. The difference is that the Q used to calculate the target and the predicted value in Q-Learning is the same Q, which means that the same neural network is used. One problem that this brings is that every time the neural network is updated, the target will also be updated, which will easily cause the parameters to not converge. Recall that in supervised learning, the label label is fixed and will not change with the update of the parameters.

Therefore, DQN introduces a target Q network on the basis of the original Q network, that is, the network used to calculate the target. It has the same structure as the Q network, and the initial weights are also the same, except that the Q network is updated every iteration, while the target Q network is updated every once in a while.

$$\varphi \leftarrow \varphi - \alpha \sum_{i} \left(s_i, a_i\right) \left[Q_{\varphi}\left(s_i, a_i\right) - \left[r\left(s_i, a_i\right) + Q_{\varphi}\left(s_i', a_i'\right)\right]\right] \tag{1}$$

In summary, compared to Q-Learning, DQN has made improvements: one is to use a convolutional neural network to approximate the behavior value function, one is to use the target Q network to update the target, and the other is to use experience playback Experience replay.

[1]Department of Computer Science, Georgetown University, District of Columbia, USA. Correspondence to: Han Jiang <hj279@georgetown.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

1.2. Proximal Policy Optimization (PPO)

The PPO algorithm is a Policy Gradient algorithm. The Policy Gradient algorithm is very sensitive to the step size, but it is difficult to choose a suitable step size. If the difference between the old and the new strategy changes during the training process is too large, it is not conducive to learning. PPO proposes a new objective function that can be updated in multiple training steps in small batches, which solves the problem that the step length is difficult to determine in the Policy Gradient algorithm. In fact, TRPO is also to solve this idea, but PPO is easier to solve than TRPO.

$$\begin{aligned} J\left(\theta\right) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} min \Bigg(& \frac{\pi_{\theta}\left(a_{it}|s_{it}\right)}{\pi_{\theta_{old}}\left(a_{it}|s_{it}\right)} \hat{A}_{it}, \\ & clip\left(\frac{\pi_{\theta}\left(a_{it}|s_{it}\right)}{\pi_{\theta_{old}}\left(a_{it}|s_{it}\right)}, 1-\epsilon, 1+\epsilon\right) \hat{A}_{it} \\ & -c_1 \left(\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{it'}\right) - V_{\theta}\left(s_{it}\right)\right)^2 \end{aligned} \tag{2}$$

1.3. Dyna

Because it is impossible to accurately and perfectly fit the real environment, the effect of pure model-based reinforcement learning is often very poor. The results can be optimized to a certain extent by combining model-based + non-model-based reinforcement learning, which is the Dyna algorithm framework (Figure 2.).

It learns both in the model and in the interaction. That is, in each iteration of the Dyna framework, it will first interact with the environment and update the value function and strategy function, and then carry out the simulation prediction of the model n times, and update the value function and strategy function as well. In this way, the experience of interacting with the environment and the prediction of the model are used at the same time.

2. Experiment Setup

2.1. Experiment Environment

Two discrete action space environments and two continuous action space environments from PyBullet are selected in the

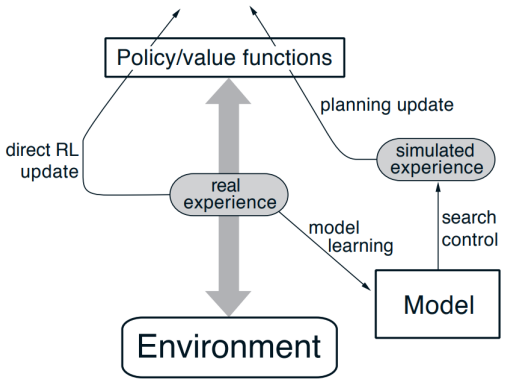

Figure 1. The general dyna structure

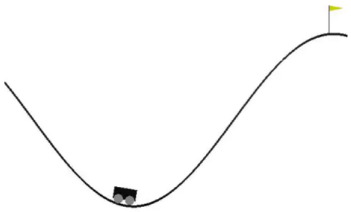

Figure 2. MountainCarContinuous-v0

experiment: CartPole-v1, MountainCar-v0, AntBulletEnv-v0, MountainCarContinuous-v0 (the first two are discrete and the last two are continuous). They all have different observation space and action space as shown below:

Environment	Obv_dim	Act_dim
CartPole-v1	4	2
MountainCar-v0	2	3
AntBulletEnv-v0	2	1
MountainCarContinuous-v0	28	8

Take MountainCarContinuous-v0 environment as an example, the environment simulates an underpowered car climbing a one-dimensional hill to reach the goal. The action (engine power) is allowed to be a continuous value. The target is on the top of the hill on the right side of the car. If the car arrives or exceeds, the episode ends. On the left, there is another mountain. Climbing this hill can be used to gain potential energy and accelerate towards the goal. On the top of this second mountain, the car cannot exceed the position equal to -1, as if there is a wall. There will be no penalty for reaching this limit (possibly in a more challenging version).

The reward is 100 for reaching the target on the hill on the right, minus the sum of the squared actions from the start to the target. This reward function presents an exploration challenge, because if the agent does not reach the goal as quickly as possible, it will find that it is best not to move and no longer find the goal.

2.2. DQN

Deep Q learning draws on the advantages of reinforcement learning and deep learning. It inherits the powerful mathematical proof of reinforcement learning and the powerful coupling capability of DeepLearning. In DQN, the Q value function is abstracted as a neural network. By sampling from the playback buffer, the Q network can be updated accordingly to obtain higher returns.

However, DQN is only known for its processing power in discrete environments. When encountering continuous environments, the initial structure cannot deal with them. Therefore, in order to make it work, data quantization is implemented for discrete and continuous input. In data quantization, multiple bins are used to assign true continuous value indexes in discrete data. By doing this, DQN can be fed with continuous values after conversion. During the given action, the output of the DQN also needs to be converted back to a continuous value quantized by the data.

The hyper-parameters of DQN in this experiment are set as shown below:

Hyper-parameters	Values
Number of trajectories	2000
Epsilon-greedy	0.1
Frozen steps	500
Update frequency	20
Buffer size	500

2.3. PPO

Similar to the idea of Trust Domain Policy Optimization (TRPO), PPO also constructs a ”trust region” to update policies. However, PPO does not use second-order derivatives, but applies first-order approximations and clipping functions to clamp the difference between the old and new strategies. By doing so, compared with TRPO, PPO can obtain competitive (and sometimes even better results) with fewer computing resources. In order to make it happen in both continuous and discrete environments, different distributions, corresponding classifications and normal distributions are applied.

The hyper-parameters of PPO in this experiment are set as shown below:

Hyper-parameters	Obv_dim
Epochs	500
Number of trajectories	10
Gamma	0.99
Alpha (learning rate)	0.0001
Beta (learning rate)	0.001
Lambda	0.95
Epsilon (clamp function)	0.2
c1 (clamp function)	0.5

2.4. Dyna-Q

In this experiment, the cost of frequent interaction with the environment is huge. However, in order to obtain better training results, a lot of conversion is required. Therefore, Dyna-Q is used to simulate the environment. The training process is mostly similar to DQN, but Dyna-Q also needs to update its environment model and complete the planning, that is, to explore the simulated environment and update the agent through the conversion of exploration. Similar to DQN, in order to be able to handle discrete and continuous environments, data quantization is also used in Dyna-QLearning.

The hyper-parameters of Dyna-Q in this experiment are set as shown below:

Hyper-parameters	Values
Number of trajectories	2000
Epsilon-greedy	0.1
Frozen steps	500
Update frequency	20
Buffer size	500
Exploration steps	10

2.5. Dyna-PPO

The hyper-parameters of Dyna-PPO in this experiment are set as shown below:

Hyper-parameters	Obv_dim
Epochs	500
Number of trajectories	10
Gamma	0.99
Alpha (learning rate)	0.0001
Beta (learning rate)	0.001
Lambda	0.95
Epsilon (clamp function)	0.2
c1 (clamp function)	0.5
Exploration steps	10

3. Experimental Results

3.1. Effectiveness

For all the figures of the result, x-axis represents epochs and y-axis represents undiscounted return.

- In Cartpole-v1 environment, PPO has the best results, converges fast and stable, while algorithm with dyna structure are less effective (Figure 3.).
- In MountainCar-v0 environment, PPO derives the most stable performance, DQN makes the largest progress. Dyna-Q has the same effectiveness but less stability (Figure 4.).

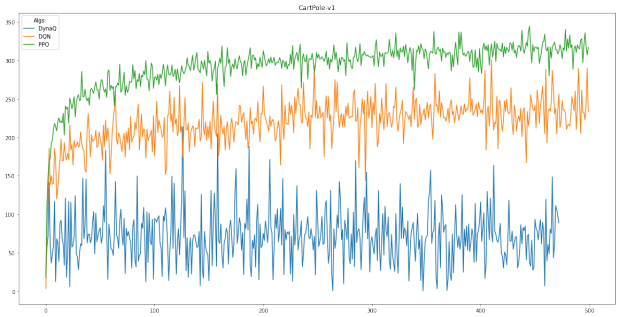

Figure 3. Cartpole-v1 result

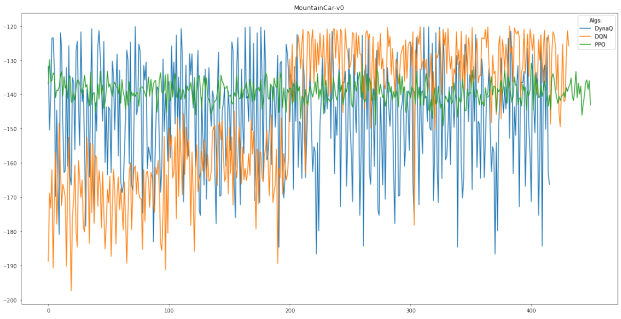

Figure 4. MountainCar-v0 result

- In MountainCarContinuous-v0 environment, PPO has the best results and gets to the good results very fast, and DQN is at the second place (Figure 5.).
- In AntBulletEnv-v0 environment, all algorithms has similar performance (Figure 6.).

3.2. Analysis

From the result, we can observe that the environment dimension can affect the performance greatly, especially for dyna algorithms. The reason is that model-based learning can better process complicated environments. Also, the performance depends on the type of the task. For example in MountainCar-v0 task, letting the car drive to the left

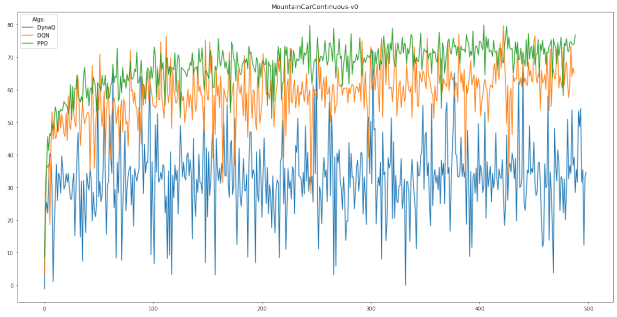

Figure 5. MountainCarContinuous-v0 result

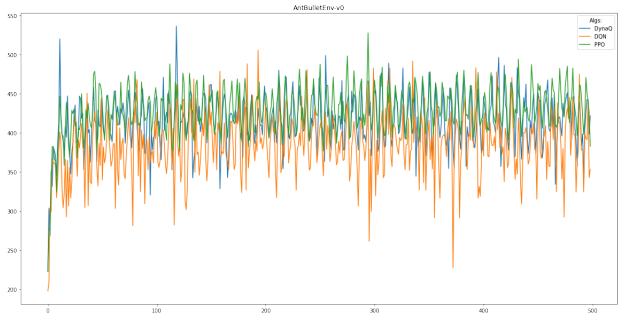

Figure 6. AntBulletEnv-v0 result

mountain is a reasonable move to speed up the car, however, PPO agent is not willing to drive the car to the left because it might end up with a negative result. Besides, the hyper-parameters can strongly impact the performance.

3.3. Efficiency

All the DRL algorithms are very time-consuming, however, I strongly believe that there is still a large room for my codes to improve, especially for PPO. Generally, PPO and Dyna-PPO runs much slower than DQN and Dyna-Q.After experiment, the most time-consuming part is calculating the advantage and discounted return.

4. Future Work and Personal Summary

Due to the time limitation and computational resources (no GPU), and most important, the coding capability, I was not able to finish the experiment of Dyna-PPO, and test the performance for more hyper-parameters. But still, the experience of this course really led me into the world of deep reinforcement learning. Since I don't have any foundation of machine learning before this semester, this course has been a really long journey for me. The most important part is that to transfer from a outsider to an expert, which requires both time (more reading and more thinking) and coding. Ashamed to say that I'm still not confident to say that I have become an expert, but the most valuable part besides the all the knowledge, is that I learned how to learn deep reinforcement learning. In the future, I will definitely start over from the beginning, implement all the essential algorithms, improve the codes that I have programmed. I strongly believe that I will become a DRL expert in a short future.

Acknowledgements

This paper is for the third homework of COSC-689 Deep Reinforcement Learning at Georgetown University, Fall 2021. I sincerely appreciate the work of professor Grace H. Yang for the impressive classes and the concern for the students both inside the class and our future development.

References

Deisenroth, Marc Peter, Carl Edward Rasmussen, and Dieter Fox. "Learning to control a low-cost manipulator using data-efficient reinforcement learning." Robotics: Science and Systems VII. Vol. 7. 2011.

Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

Segre, Alberto, and Jennifer Turney. "Planning, acting, and learning in a dynamic domain." Machine Learning Methods for Planning. Morgan Kaufmann, 1993. 125-158.