Reinforcement Learning Control Report

**1** **Background**

This paper addresses the issue of controlling an agent using Reinforcement Learning. There are five agent-environment pairs which utilize two algorithms.

To start with, we have obtained five agent-environment pairs from the OpenAI gym and classified them into two categories (Roberts, n.d.):

1) Continuous (Agent's actions can be sampled from a continuous set): Continuous Mountain Car and Pendulum.

2) Discrete (Agent's actions can be sampled from a discrete set): Mountain Car, Acrobot and Cart Pole.

In addition, we have implemented two algorithms, each intended for one category: Deep Q Network (DQN) for discrete pairs and Deep Deterministic Policy Gradient (DDPG) for continuous pairs.

Details about two algorithms can be found in *Methods* section, whilst a simple experiment based on these two algorithms and agent-environment pairs is described in *Experiment* section*.*

**2** **Methods**

2.1 Q-Learning -> Deep Q Network

Q-Learning is a value-based RL method. The value-based outputs the value of all the actions, and we choose the actions according to the highest value. Compared to policy-based RL, the value-based decision part is much more decisive and picks the ones with the highest value. It is also an off-learning temporal-difference control algorithm, because the actions executed in each episode does not reflect the actual action it will conduct, and TD update means the agent can learn when the game is processing and it will update in each step to improve the efficiency.

For Q-Learning, a Q-table is provided. Q is the action-utility function, which is used to evaluate the goodness of taking an action in a specific state. It is the memory of the agent. At each step, the table is updated according to the following formula until it converges.



Rt+1 denotes the reward acquired when taking the next step, max Q(St+1, a) denotes the maximum Q in the next state with different actions, γ is the decay factor (like a pair of glasses, to foresee the following steps), and α is the learning rate. But such a choice may cause Q to fall into local optimum. The improved strategy is ε-greedy method: each state is explored with the probability of ε.

However, in general Q-learning, when the state and action space are high-dimensional, Q-table will have too much space and state and is difficult to use. Therefore, Q-table update can be transformed into a function fitting problem here, and a function is fitted into a function to generate Q values instead of Q-table, so that similar states can get similar output actions. Therefore, we can integrate deep neural network with Q-Learning. This is the DQN algorithm. We can build a neural network with the state space to be the input layer and the action space to be the output layer. And with the processing of the neural network, we can get the Q value of the corresponding state.

There are some problems when integrating. DL is the supervised learning that requires training set. In DQN, the training labels can be represented by the reward. Recall that in Q-learning above, we used the formula to update the Q value, here we can use it as the label Q value. In addition, DL samples are independent of each other, while RL's current state value is dependent on the return value of the subsequent states. Therefore, DQN adopts the Experience Replay mechanism to store the trained data into the Replay Buffer for subsequent random sampling for training. Moreover, every time the neural network is updated, the target will also be updated, which will easily lead to parameter non-convergence. Recall that in supervised learning, the label is fixed and will not change with the update of the parameter. So, Q-targets is introduced. It is actually a mechanism to disrupt the correlation, which reduces the correlation between the current Q value and the target Q value to a certain extent and improves the algorithm stability.

2.2 Policy Gradient -> Actor Critic -> DDPG

Instead of the value-based method, policy gradient directly figures out the policy.  As the deep Q Network mentioned above, the policy gradient also used the neural network. The input of neural network is the states while the output is the actions. Initially, the agent will interact with the environment and get some reward using current parameters. The action and state from begin to the end is collect, which is called episode(or trajectory).

We calculate the discounted reward for each action, state pair. Then what we need to do is to adjust the parameters of the network, which encodes the policy, in order to maximize the reward. Since the network output is a distribution of actions and we sample the action from the distribution, we need to use the expected value as loss function. Finally, we use the gradient ascent to update the parameters.

Intuitively, if the reward of performing at is positive, then gradient ascent will increase the probability of performing this action at this state.​ On the other hand, if performing ​ at st causes the reward to be negative, the probability will be decrease.

Some techniques are used to improve the performance of the Policy Gradient. The first tip is to add baseline​. Suppose the reward for all the actions is positive and there are actions A/B/C that can be performed. In an episode, action B,C is sampled, but action A is not sampled. But now the reward for all the actions is positive, so the probability of B,C will go up while the probability of A is going down. However, A is not necessarily a bad action. Add baseline will mitigate the insufficient sample problem. Another tip is assigning suitable credits to all actions. we use the discount reward from the current action to the end of episode, rather than discount reward of whole episode.

The policy gradient has the advantage of converges to a local minimum by the hill climbing and can create the continuous action. However, there is three problems.  Firstly, the network can only update its parameter after an episode which reduce the efficiency of the algorithm. Secondly, it is difficult to decide the learning rate. If the learning rate is too large, suffer from high variance. Otherwise, the training period is too long to wait. Besides, policy gradient is on policy. The data collected for updating can only be used once.

Actor Critic is the combination of DQN and Policy Gradient, which can solve part of their problems. DQN is used in Critic to give Q-value while the Policy Gradient is used to give the action as Actor. Actor selects an action based on the probability, and the Critic evaluates the score of an action based on the Actor's action, and an Actor modifies the probability of an action based on the score of the Critic.

Compared with the policy Gradient, Actor Critic can be updated in a single step, which is faster than traditional Policy gradients. However, as Actor depending on the value judgment of the Critic, it is difficult to converge. In order to solve the convergence problem, Deep Deterministic Policy Gradient (DDPG) is proposed. DDPG use two neural networks that calculate Q. In Q target, actions are selected using Actor based on the next state. In this case, the Actor is also an Actor target (with the Actor's long-ago parameters). The Q target obtained by this method can cut off the correlation just like DQN and improve the convergence.

**3 Experiment**

You can see in the code that each agent-environment pair renders its statistics throughout execution: Success/Failure, Episode number, Task number, Episode reward. Depending on whether the pair is discrete or not, it will print out Epsilon or Variance, respectively. Both variables are related to the exploration-exploitation ratio.

The number of episodes is set to 100 for all pairs. However, the number of tasks per episode varies based on the termination condition. Moreover, the reward functions are given and explained in the code for each pair. Given these initial parameters, our experiment includes recording the first successful episode, as well as success ratio for each pair. Three trials have been conducted for every pair, and the recorded data has been given in Table 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pair | First success episode  1 2 3 avg | | | | Number of successes  1 2 3 avg | | | | Total number of episodes | Success ratio  1 2 3 avg | | | |
| Mountain Car | 5 | 3 | 3 | 3.67 | 88 | 87 | 91 | 88.67 | 100 | 88% | 87% | 91% | 88.67% |
| Acrobot | 2 | 2 | 1 | 1.67 | 98 | 91 | 98 | 95.67 | 100 | 98% | 91% | 98% | 95.67% |
| Cart Pole | 62 | 28 | 44 | 44.67 | 15 | 56 | 33 | 34.67 | 100 | 15% | 56% | 33% | 34.67% |
| Continuous Mountain Car | 22 | 24 | 23 | 23.00 | 78 | 74 | 74 | 75.33 | 100 | 78% | 74% | 74% | 75.33% |
| Pendulum | 29 | 32 | 30 | 30.33 | 56 | 55 | 56 | 55.67 | 100 | 56% | 55% | 56% | 55.67% |

*Table 1. Experimental data obtained from training*

We can observe the different first success episodes and number of successes between the pairs in the table during network training. One possible explanation is the different influence of reward function on the convergence in each algorithm and within each algorithm, the different suitability of the reward function to the agent-environment pair.

**4** **Contribution**

Rui XU *– Q-Learning -> DQN*; Implementation of DQN and DDPG

Zhiyuan ZHANG – *Policy Gradient -> Actor Critic -> DDPG;* Implementation of DDPG

Aleksa JELACA – *Background* and *Experiment;* Implementation of reward functions

**5** **Reference**

[1] Retrieved from <https://zhuanlan.zhihu.com/p/25319023>.

[2] Retrieved from <https://github.com/openai/gym/tree/master/gym/envs/classic_control>.

[3] Retrieved from <https://blog.csdn.net/weixin_40056577/article/details/106459867>.

[4] Retrieved from <https://mofanpy.com/tutorials/machine-learning/reinforcement-learning/>.

[5] Retrieved from <https://www.zhihu.com/question/26408259>.

[6] Retrieved from <https://blog.csdn.net/qq_30615903/article/details/80744083> .

[7] Roberts, D. (n.d.). *Continuous and Discrete Functions* - MathBitsNotebook(A1 - CCSS math). Retrieved April 19, 2021, from <https://mathbitsnotebook.com/Algebra1/FunctionGraphs/FNGContinuousDiscrete.html>.

[8] Hung-yi Lee. (2018, June 9). *DRL Lecture 1: Policy Gradient*. YouTube. <https://www.youtube.com/watch?v=z95ZYgPgXOY&list=PLJV_el3uVTsODxQFgzMzPLa16h6B8kWM_>.

[9] Retrieved from <https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow/tree/master/contents>.