Proximal Distilled Evolutionary Reinforcement Learning

Cristian Bodnar, Ben Day, Pietro Lio

Abstract

Reinforcement Learning (RL) has been a great success on many challenging tasks, due to the partnership with Deep Neural Net works (DNNs). Genetic Algorithms (GAs) have great advantages for solving large-scale problems. but it slowly runs out of favor due to their inability to solve more complex problems. The recently proposed Evolutionary Reinforcement Learning (ERL) framework has demonstrated the capacity of the two methods to enhance each other. And a novel algorithm called Proximal Distilled Evolutionary Reinforcement Learning (PDERL) that is characterised by a hierarchical integration between evolution and learning. However, the two appealing algorithms suffer from the problem of overestimation of Q values, which leads to slow convergence and low convergence results. In this paper, we change the DDPG network to a non-delayed update TD3 network, in addition to modifying the reward mechanism of the environment. We evaluate our algorithm in four robot locomotion environments from the OpenAI gym. Our method outperforms PDERL, in all the environments.

Keywords：Genetic Algorithms，DDPG，TD3

1 Introduction

Reinforcement learning has recently enjoyed great success by producing artificial agents that can master to play Atari games or to control robots to achieve complex tasks such as grasping objects, due to partnership with Deep Reinforcement Learning (DRL).

At the same time, Genetic Algorithms (GAs), usually seen as a competing approach to RL, have run out of favour due to their inability to scale up for evolving Deep Neural Networks (DNNs). As the recent progress in RL has shown, DNNs are required to solve complex environments. Contrary to this dichotomic view of RL and GAs, in the physical world, evolution and learning are complementary processes that continuously interact., Khadka and Tumer (2018)[1] have recently demonstrated on robot locomotion tasks the practical benefits of merging the two approaches in their Evolutionary Reinforcement Learning (ERL) framework. And Proximal Distilled Evolutionary Reinforcement Learning (PDERL) improves the genetic operator on the basis of ERL. However, the reinforcement learning network they use suffers from an overestimated Q value and incorrect Reward due to the wrong end flag in the robot locomotion environments. This paper brings the following contributions:

First, modify the single Critic network output of DDPG to the minimum of the two Critic network outputs.

Second, fixed reward feedback for the robot locomotion environments.

Third, Proposes two novel genetic operators that are based on backpropagation. These operators do not cause catastrophic forgetting in combination with simple DNN representations.

2 Related works

This section introduces the Evolutionary Reinforcement Learning (ERL) algorithm and the genetic operators it uses.

2.1 Evolutionary Reinforcement Learning

The proposed methods build upon the ERL framework proposed by Khadka and Tumer (2018)[1]. In this framework, a population of policies is evolved using GAs. The fitness of the policies in the population is based on the cumulative total reward obtained over a given number of evaluation rounds. Alongside the population, an actor-critic agent based on DDPG[2] is trained via RL. The RL agent and the population synchronise periodically to establish a bidirectional transfer of information.

2.2 Genetic encoding and variation operators

The policies in the ERL population are represented by neural networks with a direct encoding. Traditional genetic coding operates directly on the parameters of the network. Crossover is a direct exchange of parameters in the network, and variation is a random modification of them.As the number of parameters increases, it is becoming increasingly unlikely that these random weight changes produce better policies.

3 Method

3.1 Overview

As shown in Figure1, a population of policies is evolved using GAs, including crossovers and mutations. Actors in a population interact with the environment to generate experience. The experience is stored in the replaybuffer, which provides training samples for reinforcement learning. Actors can evolve through crossovers and mutations, and can also be updated through the Actor generated by reinforcement learning. In addition, the envorment reward has been modified to better represent the state of the environment. And the output of Critic in DDPG has been change into the minimize value of the two different networks, which can effectively address the problem of overestimate Q value.

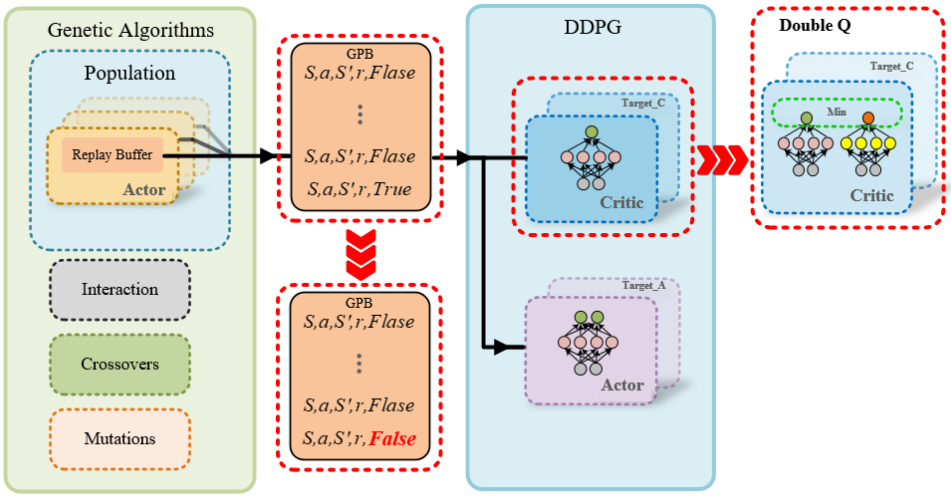

Figure 1: Overview of the method

3.2 Q-Filtered distillation crossovers

Unlike traditional genetic operators, Q-Filtered distillation crossovers can effectively inherit the advantages of the parent without causing catastrophic forgetting.

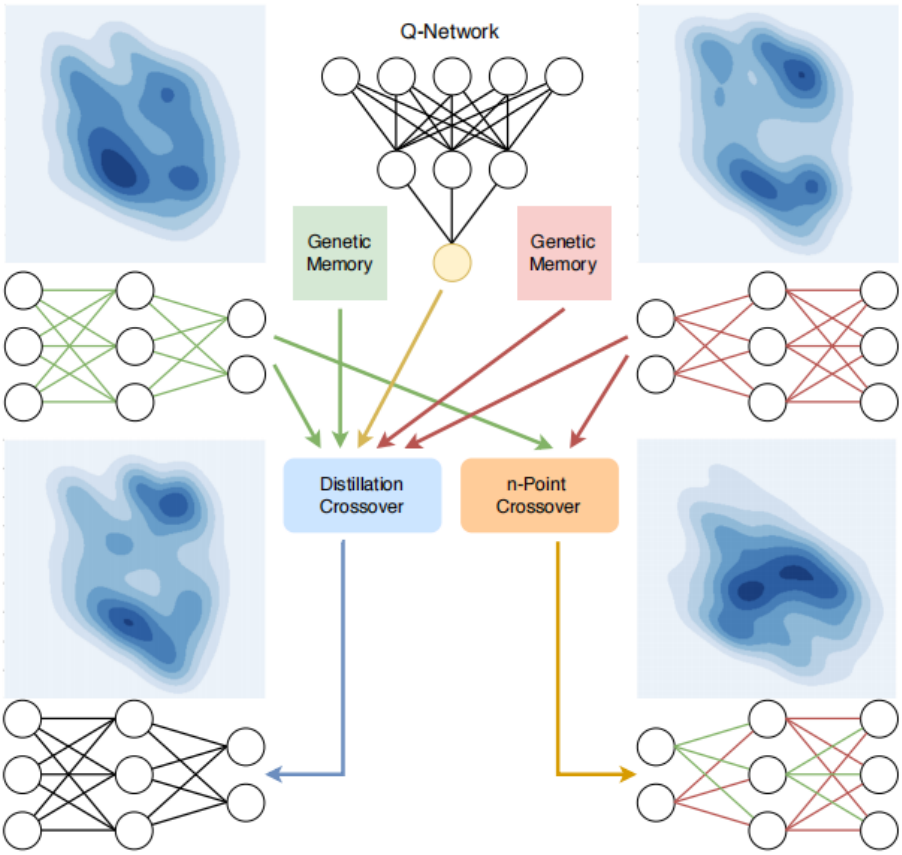

Figure 2: Q-filtered distillation crossover compared to npoint crossover. The contour plots represent the state visitation distributions of the agents. The Q-filtered distillation crossover selectively merges the behaviours of the two parents, whereas the modes of the state visitation distribution obtained by the traditional crossover are disjoint from the modes of the parent distributions.

3.3 Proximal mutations

In contrast to the traditional variation operator, Proximal mutations can be more effective in avoiding catastrophic forgetting, and we take the gradient of backpropagation as the important degree that determines the size of the variation of the net parameters in the population, changing the variation in a way that well preserves the fitness of the parents and also increases the diversity of the population

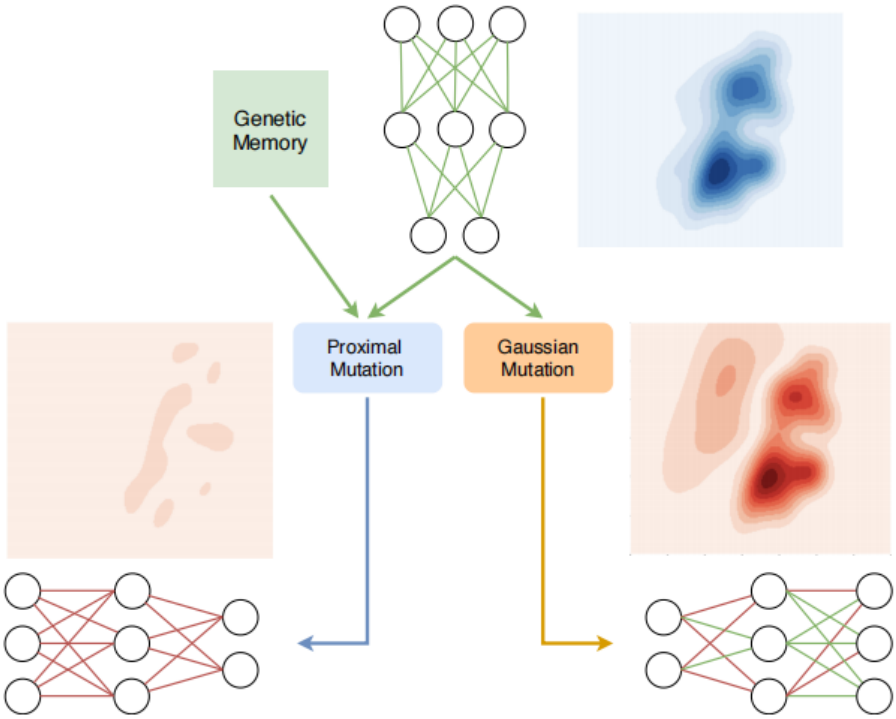

Figure 3: Proximal mutations compared to Gaussian mutations. The blue contour plot shows the state visitation distribution of the parent policy. The red contour plots show the difference between the distribution of the children and that of the parent. The behaviour of the policy obtained by proximal mutation is an adjustment to the parent behaviour. In contrast, the traditional mutation produces a divergent behaviour. At the same time, the Gaussian mutation modifies only a fraction of the weights (shown in red).

4 Implementation details

4.1 Comparing with released source codes

This work is based on the source code given by the author with the following two improvements：

First, modify the single Critic network output of DDPG to the minimum of the two Critic network outputs.

Second, fixed reward feedback for the robot locomotion environments.

Firstly, The problem of DDPG is overestimate for the Q value which causes agent make wrong judgments about the environment.

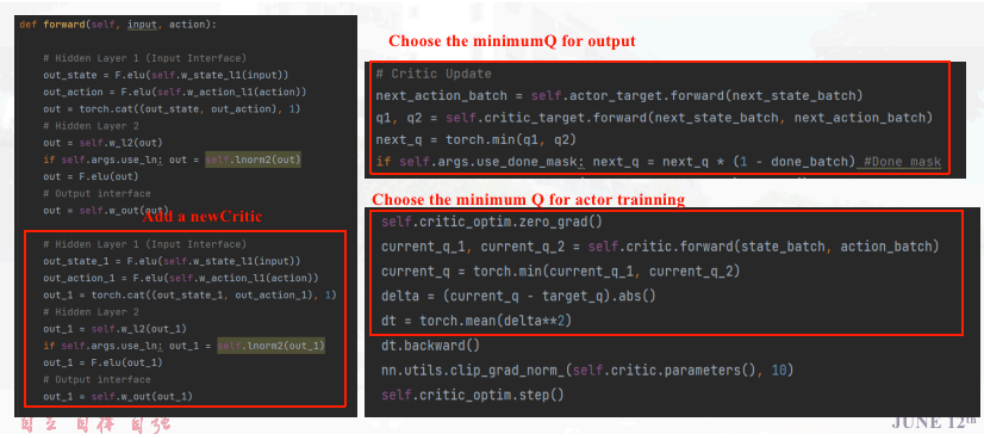

Figure 4: DDPG code modification

As shown in Figure4, I added a Critic network with the same structure as the original Critic network but with different parameters. I replace the minimum Q value of the two Critic networks with the Q value of the

original single network, so that I can get a more accurate Q value.

Secondly, The actions in the locomotion environments are often not stopped, and the environments take a certain number of steps as the termination state of the environment, but this affects the agent's estimation of the value of the environment, so I make the following improvements to the code.

```
def evaluate(self, agent: ddpg.GeneticAgent or ddpg.DDPG, is_render=False, is_action_noise=False,
             store_transition=True, net_index=None):
    total_reward = 0.0
    total_error = 0.0

    state = self.env.reset()
    done = False
    times = 0
    while not done and times < 1000:
        times = times + 1
        if store_transition: self.num_frames += 1; self.gen_frames += 1
        if self.args.render and is_render: self.env.render()
        action = agent.actor.select_action(np.array(state))
        if is_action_noise:
            action += self.ounoise.noise()
            action = np.clip(action, -1.0, 1.0)

        # Simulate one step in environment
        next_state, reward, done, info = self.env.step(action.flatten())
        if times == 1000:
            done = False
        total_reward += reward
```

Figure 5: Environments feedback code modification

Different end flags affect the calculation method of Q value, and the modified code can get a more accurate Q value, so it can achieve better results.

4.2 Experimental environment setup

Operating System is ubuntu 18.04，GPU is GTX 3090，Create a virtual environment using anaconda and install the relevant dependencies in requirements. In addition, mujoco210 must be install to interact with agent.

5 Results and analysis

The experimental results show that the algorithm can solve all problems effectively in all environments, and the convergence points are better than the original results, and the speed of convergence is greatly improved in this scenario of Walker2d-v2

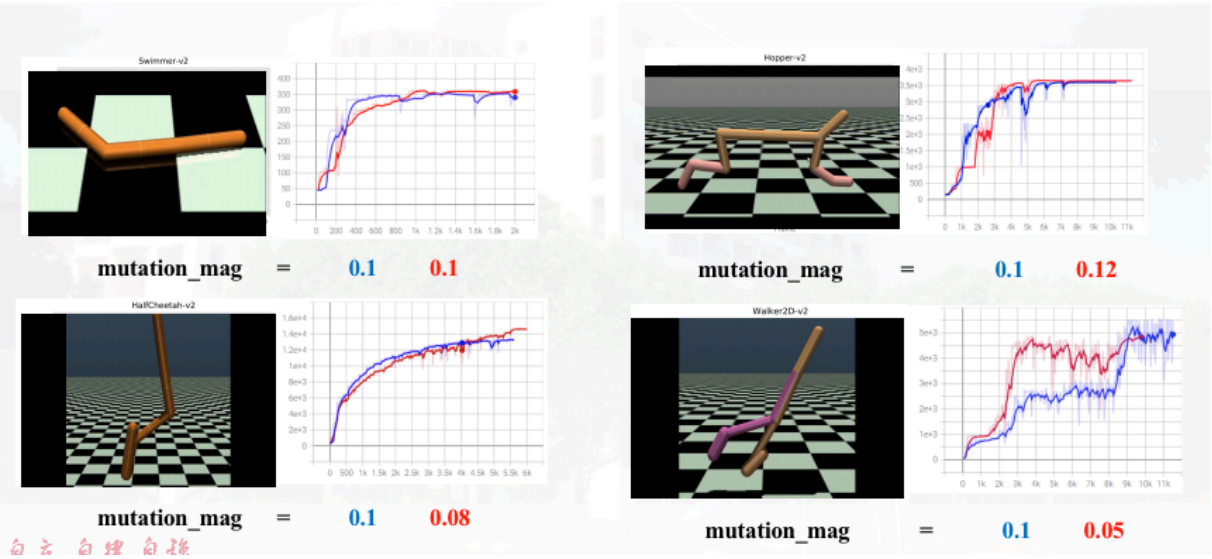

Figure 6: Comparison of source code results (blue) and modified code results (red)

The only parameter adjusted in this experiment is mutation_mag, which represents the mean value of the Gaussian distribution at the time of mutation, with larger values representing more mutations.

6 Conclusion and future work

The most advanced algorithm to solve the problem of excessive Q is SAC, so the method still has a lot of room for improvement, and secondly, the parameter mutation_mag is not adjusted very sufficiently due to the arithmetic power, and there may be some better parameters for the results.

References

[1] KHADKA S, TUMER K. Evolution-Guided Policy Gradient in Reinforcement Learning[J]., 2018.

[2] LILLICRAP T P, HUNT J J, PRITZEL A, et al. Continuous control with deep reinforcement learning[J]. Computer ence, 2015.