Deep Reinforcement Learning Algorithm Notes

YOU Jiang

July 31, 2019

Contents

1 Introduction

One of the primary goal in the field of artificial intelligence is solve complex task from unprocessed, high dimentional, sensory input[2]. The Deep Reinforcement Learning algorithms designed by google scientist(DDPG) showed its power in playing Atari video games and the traditional game GO. However, a major obstacle facing deep RL in the real world is their high sample complexity.[1] To respond to this challenge, the article[1] provided an efficient learning strategy, prioritised experience replay, to learn from samples with high priority.
In the previous project, we redesigned the experience replay buffer to accelerate the learning process. We've seen that the prioritised experience algorithm could find an optimal solution in a limited steps. However, the implemented algorithm is not stable, it fails occasionally for some unknown reason. In order to improve our previous result, in this article, we present the neuron network written with the latest released TensorFlow library (at Feb 2018). At last, the experience showed that in the mountain car environment the strategy converges to the best solution.

2 Algorithm

The deep reinforcement learning algorithm consists of 2 neuron networks, Actor and Critic. More precisely, the Actor net takes the state as input and gives a most appropriate action as output.

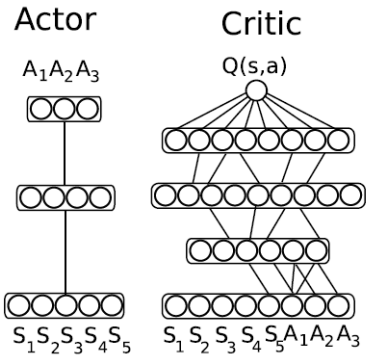

Figure 2.1: The Actor and Critic nets [4]

In the mean time, the Critic net evaluates the state and its related best action and it produces a scalar value to show its confidence of this state-action couple. The loss function in deep reinforcement learning algorithm is different from the other type of neuron networks. In stead of using the mean square error between prediction and labels, it minimizes the error of prediction between the current step and next step.
The critic on current step is approximately the critic on the next step plus the current reward:

$$y_i = r_i + \gamma max_a Q(s_{i+1}, a|\theta)$$

It means that, if the agent knows that at step s_{i+1} it would have a score $max_a Q(s_{i+1}, a|\theta)$, then it would estimate the score at current step as y_i, the sum of current reward r_i and the next score. (The γ is the discount factor).
As discussed above, the loss function is the error between the scores at current state and the next:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta))^2$$

, where s_i is current state and a_i is current action. It use a mean definition because the algorithm use a minibatch of size N to train the model.
To update the Actor net, the algorithm take use of the critics of consecutive states. With the gradient of critic on an action, we could apply them on the the Actor net to guide the training. For example, if the critic $y_{i+1} > y_i$, it means the estimation of score at the state s_{i+1} is better than that at state s_i, and it is better to execute the action to move the agent from state s_i to $si+1$.

3 Implementation

The implementation of deep reinforcement learning is based on Tensorflow v1.14.0, where high level api had been introduced. Fortunately, we could take the advantage of avoiding programming on neuron level and concentrate in the layers and structures.
In the previous experience, the agent gets stuck usually at the valley or the left wall for unknown reason. After checking the output of the critic net, I found out that the poor stability was caused by the learning rate and the forget catastrophe. A high learning rate causes divergence while a low learning rate make the training process incredibly long. As a response to this issue, I used the descending learning rate so that the convergence comes definitely, and it converges stably to the best strategy with a high possibility. The extra buffer stores only the samples from an episode successfully executed and better than all previous executions. It helps the agent not to overwrite on (not to forget) the important samples. I added also batch normalization layer to make the training more stable.

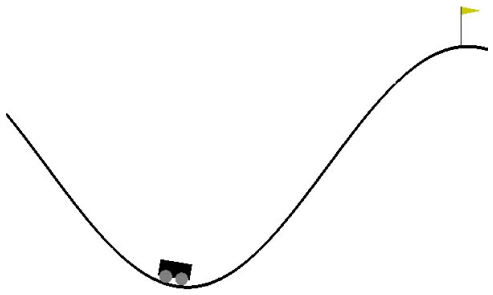

Figure 3.1: Mountain car version continue, source: http://gym.openai.com

4 Experience

The experience has been executed in the environment "Mountain-Car-Continuous-v2" for 10 time with different size of hidden layer.

Paramètres	Acteur	Critique
hidden layer size	50, 100	50, 100
replay buffer size	40000	
minibatch size	64	
α	0.001	0.0003 -> 0.00001
γ	0.99	
α of power law	-	0.6 -> 0.01

At each execution, the agent plays at most 300 episodes and it converges usually at the 50th episode. In conclusion, the training of this design is fast and the performance is stable.

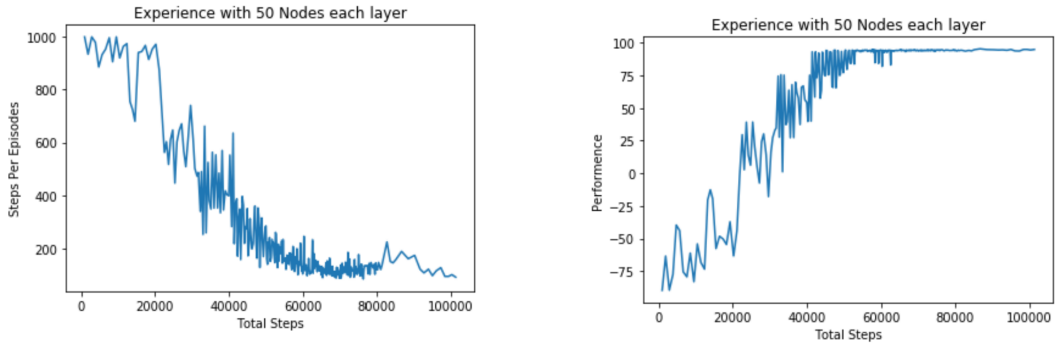

Figure 4.1: The experiments with 50 nodes at each hidden layer

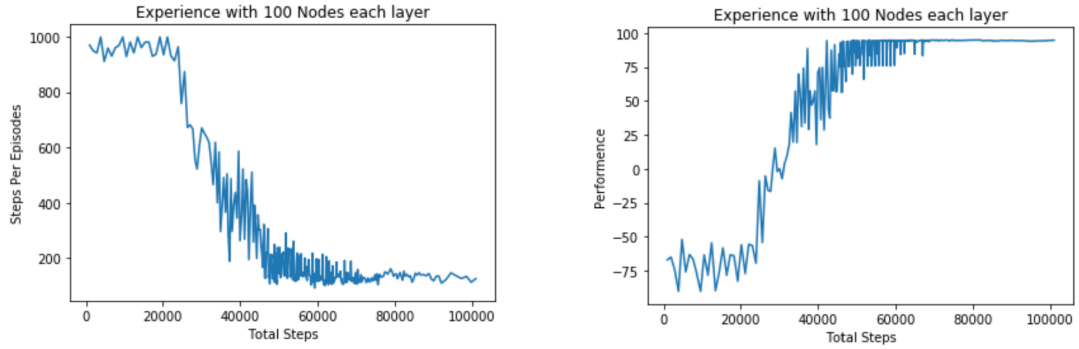

Figure 4.2: The experiments with 100 nodes at each hidden layer

In the classical reinforcement learning, the Q-matrix stores the Q-values of all possible states. Based on this idea, we output the critic values on the discrete couples of state: (Position, Velocity), and also the converged strategy(acceleration) on the states.

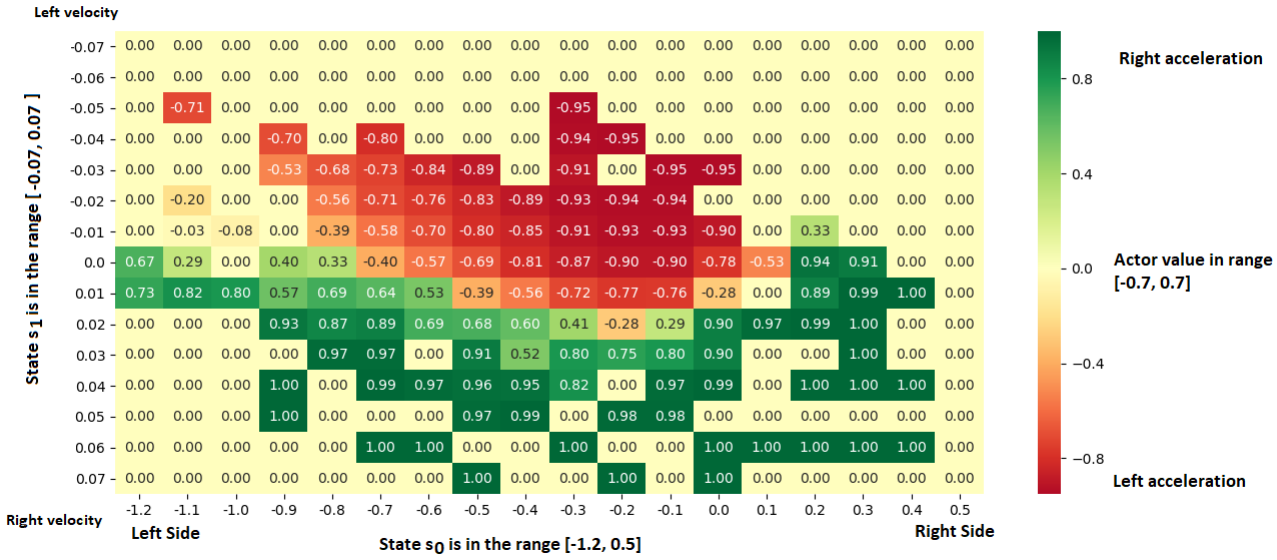

Figure 4.3: The Actor net output values at the 50th episodes

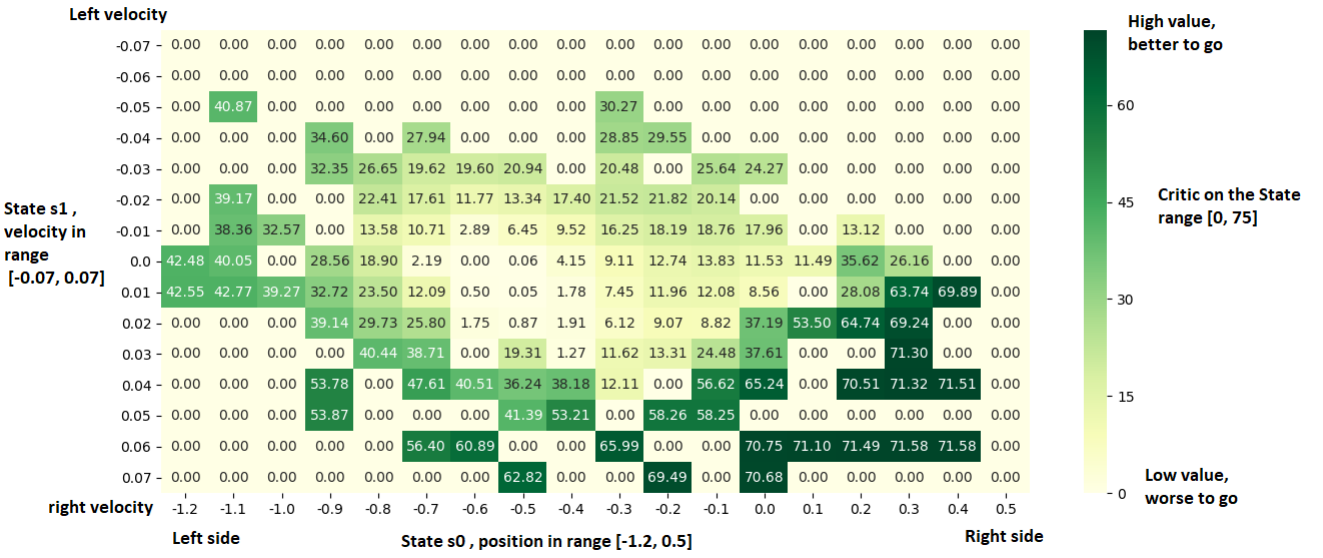

Figure 4.4: The Critic net output values at the 50th episodes

This proves that our network could find a solution starting from any states in this enviroment. Specifically, the best strategy shows that the car goes a lit bit right then left to gather enough energy, then it shifts to the end point. The best strategy is observable from the matrix.

5 Further Work

This experiment is a short term work(within 2 weeks) and we could add more interesting tests in the future based on this implementation, for examples, executing with different setup of parameters and different designs to understand better the function of parameters.
Since 2016, multiple articles tried to improve the performance of DDPG, such as DDQN, Dueling DDQN, A3C, Distributional DDQN, Noisy DDQN[3], I would test them in the future works.

References

[1] Tom Schaul, John Quan, Ioannis Antonoglou and David Silver *Prioritized experience replay*, 2016

[2] Timothy P. Lillicra, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess,Tom Erez, Yuval Tassa, David Silver Daan Wierstra *CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING*, 2016

[3] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver *Rainbow: Combining Improvements in Deep Reinforcement Learning*, 2018

[4] Olivier Sigaud, *Deep Reinforcement Learning Algorithms*, 2016

[5] Arnaud de Broissia, Olivier Sigaud, *Study of a deep reinforcement learning algorithm*, 2016

[6] Simon Ramstedt, *Deep Reinforcement Learning for Continuous Control*, 2016

[7] Schaul, T., Quan, J., Antonoglou, I., Silver, D. *Prioritized experience replay*, 2015

[8] Mnih, et al. *Human-level control through deep reinforcement learning*, 2015

[9] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra *Continuous control with deep reinforcement learning*, 2015

[10] Ioffe S. Szegedy C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift*