Exercise 6

Breno Aberle - 876438
ELEC-E8125 - Reinforcement Learning

November 2, 2020

1 Task 1

For Task 1 we needed to implement the forward(), update-policy(), and get-action() function. The most difficult was by far the implementation of the update-policy function. It was difficult to decide which variables are going to be detached and to use self.gamma in critic-loss or not. From the slides is was not really clear if we should use gamma in that case or not. And there was also room for speculation, where to detach. The variance of the plots makes it really difficult to identify which constellation is correct. Because sometimes there are good results and sometimes it's bad and due to variance it is not easy to identify what code was correct. But after the discussions on slack, the literature, and online research I ended up working results. The performance plot for Task 1 can be found in figure 1.

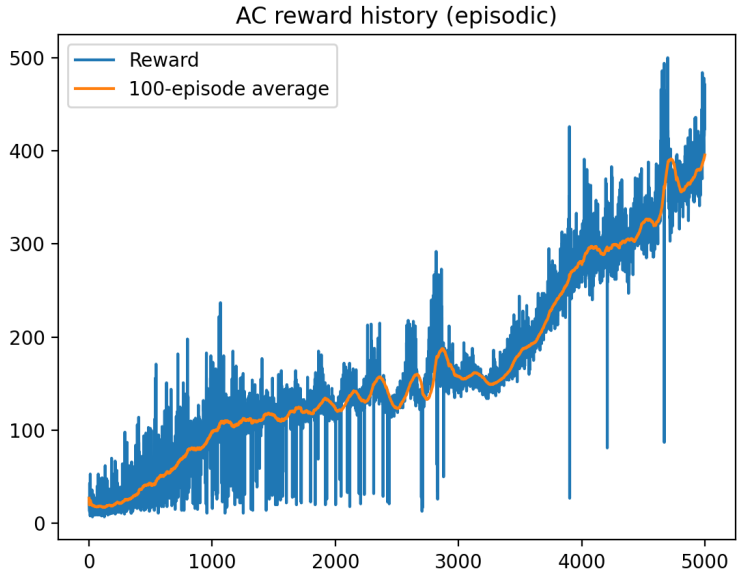

Figure 1: Performance plot Actor-critic with policy update after the end of the episode.

2 Question 1

What is the relationship between actor-critic and REINFORCE with baseline?

Actor-critic is a special form of REINFORCE. The difference is what feedback the actor uses to change its policy. In REINFORCE we use return (total reward) as our target. In Acotr-critic we define a Critic, that tries to predict the long term value of a state V(s). The benefits are low variance, because now we can update our parameters at each step using the Q-values. That also ensures faster convergence, but at the expense of initial bias towards however the model was initialised. So we are increasing bias while reducing variance. However, in RL bias can be handled better than variance, and therefore it is worth it. We can also use non-episodic policy updates, so we don't need to wait till the episode finishes.

3 Question 2

How can the value of advantage be intuitively interpreted?

The value of advantage is a measure of how much is a certain action a good or bad decision given a certain state. It states what is the advantage of selecting a certain action from a certain state and it tells about the extra reward that could be obtained by the agent by taking that particular action.

4 Task 2

For task 3, instead of updating the policy after the end of every episode, the update of the policy is now done after every 50 timesteps. The performance plot is illustrated in figure 2. To be fair, I want to mention at that point that the plots vary a lot. After every rerun, the plots may look different. Sometimes smaller differences and sometimes bigger differences. But according to the discussions on slack that phenoma is normal and occurs for everyone.

5 Task 3

In task 3 we use now parallel data collection. We basically just needed to run the parallel-cartpole.py script, if the code of the previous tasks is working properly. The performance plot for this task is displayed in figure 3.

6 Question 3

How is parallel data collection different from the parallelism in multiple-cartpoles.py script we've seen in Exercises 1 and 5?

In multiple-cartpoles.py every agent has its own environment, acts in that environment, and makes updates that only affect that environment. It doesn't interfere with the other agents. However, in parallel data collection the agents share a network and update a common environment. That leads to better optimization and usage of more diverse training data.

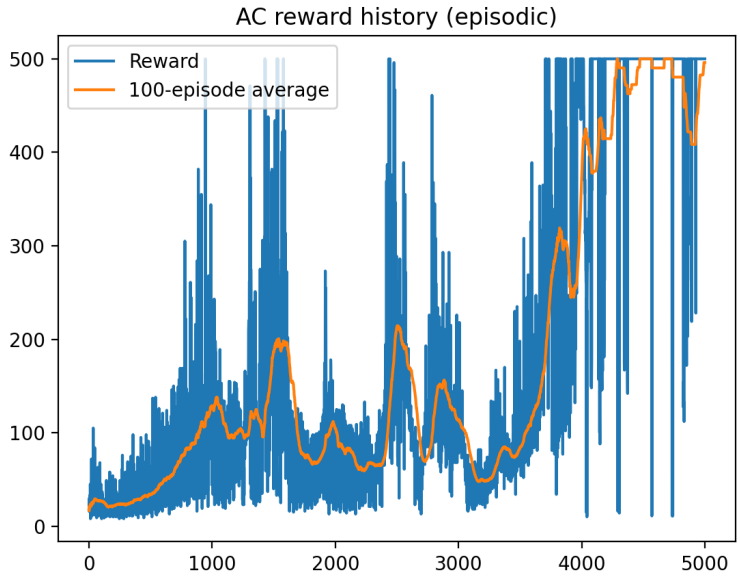

Figure 2: Performance plot Actor-critic with policy update after every 50 timesteps.

Can it replace multiple runs of the training algorithm for comparing RL algorithms?

Parallel data collections can't really replace multiple runs of the training algorithm for computing RL algorithms. Since we use a common environment in parallel data collections, we see only how the algorithm performs for that environment. However, the purpose of multiple runs was to see how the algorithm performs performs in different environments and average of the variance. You have to have independent environments to compare the algorithm.

7 Question 4

In terms of initial performance, REINFORCE seems to completely outperform all tested A2C flavours on Cartpole, despite being a simpler algorithm. Why is it so?

The first reason is that Reinforce has much bigger variance compared to others A2C. Therefore, more exploration will be done and consequently it will converge faster. However, the gradient updates for the function approximation have normally relatively low step size. That means it takes several iterations till it converges to good performance.

8 Question 5.1

How do actor-critic methods compare to REINFORCE in terms of bias and variance of the policy gradient estimation? Explain your answer.

Reinforce has high variance and low bias, while actor-critic has low variance and high bias.

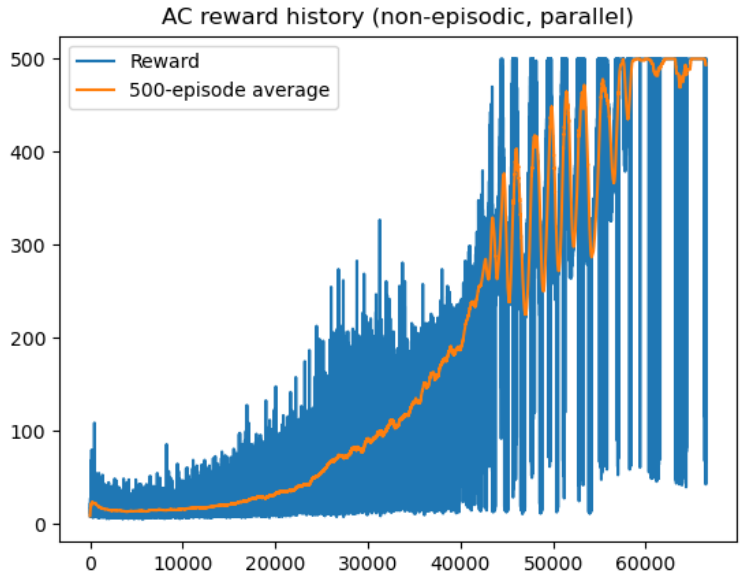

Figure 3: Performance plot Actor-critic with parallel data collection.

Variance makes the gradient instable which results in worse updates - doesn't converge to optimum and therefore worse results in the long run. How variance allows faster exploration to converge faster to optimum. In the long-run actor critic algorithms have better performance. low variance allows better exploitation. However, it takes longer to converge.

9 Question 5.2

How could the bias-variance tradeoff in actor-critic be controlled?

One approach to control the bias-variance tradeoff in actor-critic is to implement the asynchronous actor-critic [1]. Some other popular RL algorithm use a single agent and a single environment. Asynchronous (A3C) uses multiple agents with each agent having its own network parameters and a copy of the environment. Each agent is controlled by a global network and each agent contributes with updates to the total knowledge of the global network. With a global network each agent has access to more diversified training data. In real time human beings learn also from other humans while being in the same environment.

10 Question 6

What are the advantages of policy gradient and actor-critic methods compared to action-value methods such as Q-learning? Explain your answer.

In Q-learning, we want to learn a single deterministic action from a discrete action space by finding the maximum value. Q functions can be implemented from simple discrete table. That provides guarantees of convergence, but this does not guarantee to result in near-

optimal behavior. Policy gradient can't be directly applied to discrete space because you need to be able to calculate gradients. However, for instance with importance sampling you can find a workaround. Q-Learning is often much faster to learn a policy because we are using a discrete state space. PG/AC on the other hand need to sample from the environment in order to evaluate progress. Q-learning can also use experiences collected from previous policies and is off-policy.

Policy gradients and Actor critic aim to learn a map from state to action, which can be stochastic, and works in continuous action spaces. Large and continuous action space are possible. We are using a stochastic policies and trajectories have to be sampled from the current policy. Thus, policy gradient methods are on-policy methods.

References

[1] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.