Policy Gradient Methods Cart Pole

Francesco Paolo Carmone
s308126
Computer Engineering in Automation and Cyber-Physical Systems
Politecnico di Torino
`francescopaolo.carmone@studenti.polito.it`

Task 1

The following graphs and results have been obtained by changing the way the estimated losses were calculated. Once obtained the model, I tested it three times and reported the results below.

In task 1a, used for benchmark, the new parameters have been calculated as

$$\theta_{t+1} = \theta_t - \alpha\gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta_t) \tag{1}$$

where α is the (ADAM) optimizer learning rate, G_t the total return, and θ_t the current parameter.

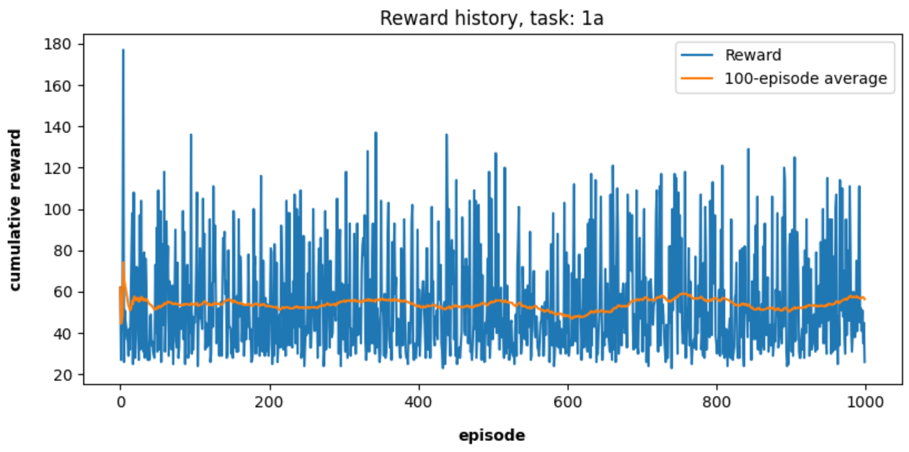

Figure 1: Basic REINFORCE without baseline

```
1a
Average test reward: 52.87 episode length: 52.87
Average test reward: 51.28 episode length: 51.28
Average test reward: 50.83 episode length: 50.83
```

Then we test a new model with a constant baseline, $b = 20$, whose update rule is:

$$\theta_{t+1} = \theta_t - \alpha\gamma^t (G_t - 20) \nabla \ln \pi(A_t|S_t, \theta_t) \tag{2}$$

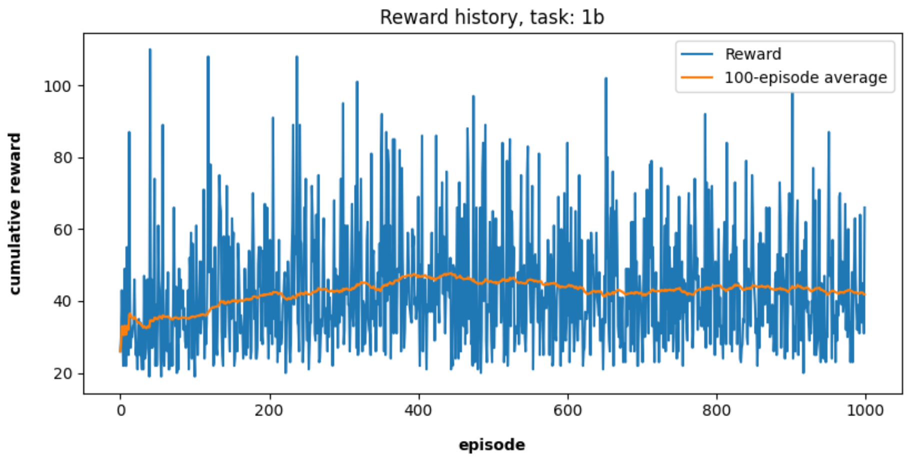

Figure 2: REINFORCE with constant baseline $b = 20$

```
1b
Average test reward: 67.24 episode length: 67.24
Average test reward: 65.0 episode length: 65.0
Average test reward: 73.87 episode length: 73.87
```

We can see improvement with respect to the previous results, justified in Question 1. Finally for task 1c, after calculating G_t, I normalized it to zero mean and unit variance

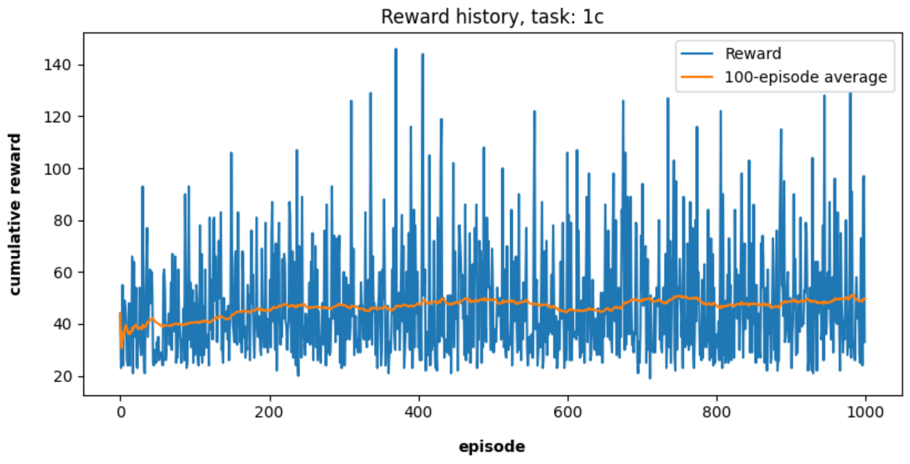

Figure 3: REINFORCE with discounted rewards normalized to zero mean and unit variance

```
1c
Average test reward: 35.56 episode length: 35.56
Average test reward: 38.2 episode length: 38.2
Average test reward: 35.88 episode length: 35.88
```

And the results are frankly disappointing.

Question 1.1 & 1.2

How would you choose a good value for the baseline? How does the baseline affect the training?

Adding a baseline leaves the mean value of the gradient unchanged, but effects the variance. It can be chosen arbitrarily as long as it doesn't vary with the action a. It impacts the speed of learning and for that reason I would choose a baseline that could reduce significantly the variance. Since the state values assume different values, a natural choice for the baseline is the estimate of the state value, so that their difference can reach zero. Although

Real-world control problems

Question 2.1

What could go wrong when a model with an unbounded continuous action space and a reward function like the one used here (+1 for survival) were to be used with a physical system?

There's an infinite amount of actions that will lead to the same result. This modified environment allows unbounded actions in the form of real values that indicates an arbitrary force. The chosen action will then act on the environment through an actuator, which is an electro-mechanical device with finite capacities: not every action, or *command*, can be satisfied by the actuator, as it will saturate. Let's assume that this saturated action results in a positive reward; the same can be said about any another action that saturates. To be precise, since we're dealing with real numbers, there's an infinite amount of them, so there's an infinite amount of actions that will result in an equally positive reward. Even if we were working with an ideal actuator that doesn't saturate, the exploration of the action space becomes challenging if not impossible, the algorithm will have to explore an infinite range of actions to find the optimal ones. Once found them, they might be sparse, and it would be hard to find an optimal policy. Unlike in simulations, in a real environment every action our agent takes has an effect on it. Exploring too many actions could simply be unfeasible or unsustainable: can we, for example, afford to lose autonomous vehicles while exploring the action space every time it crashes?

Question 2.2

How could the problems appearing in Question 2.1 be mitigated without putting a hard limit on the actions? Explain your answer.

A different approach to solve the problem I mentioned earlier is instead to learn the statistics of a probability distribution for the actions, for example a Gaussian, where the mean $\mu(s, \theta)$ and the standard deviations $\sigma(s, \theta)$ are given by the parametric function approximators that depend on the state, as suggested in the book (1). The mean can be approximated with a linear function $\mu(s, \theta) \doteq \theta_\mu x_\mu(s)$, and the standard deviation with the exponential of a linear function $\sigma(s, \theta) \doteq \exp \theta_\sigma^T x_\sigma(s)$. Another approach could be bootstrapping from another method that has already learned some basic skills, and learned to avoid some actions entirely.

Discrete action spaces

Question 3

Can policy gradient methods be used with discrete action spaces? Why/why not? Which steps of the algorithm would be problematic to perform, if any? Explain your answer

Yes, policy gradient methods can be used with discrete action spaces as well, and a common parameterization is to form parameterized numerical preferences $h(s, a, \theta)$ for each state-action pair. That function can be chosen arbitrarily for example as an exponential soft-max distribution, or can computed by a neural network. The only problem I can think about is discretizing the action space with just few actions will inevitably result in a loss of precision if compared to a continuous action space, as it limits the amount of values an action can assume.

Task 2

Both PPO and SAC are Actor-Critic methods as they share its defining structure of such methods, and their differences consist in the tecniques used for updating the policy and the value state. SAC utilizes entropy maximization, encurages exploration, and uses a soft value function for finding an optimal trade off between maximazing the expected return and the maximized entropy. PPO ensures that its policy update are performed in a conservative manner, by preventing policy changes too large or too small, trough a clip function.

Question 4

Analyze the performance of the trained policies in terms of reward and time consumption. Additionally, compare the results with the REINFORCE algorithm you have previously obtained, highlighting any notable differences in terms of learning stability and convergence speed. Motivate with some hypotheses all the results obtained.

The following table has been compiled by averaging out the results of three independent execution of the model described by the parameters in each row, and fixing the target timesteps to 25.000.

PPO, $25k$ timesteps

γ	time	μ	σ
0.003	37	167.89	64.662
0.03	47	165.9	50.46
0.3	36	278.77	82.60

SAC, $25k$ timesteps

γ	time	μ	σ
0.003	813	430.85	47.79
0.03	661	394.14	84.71
0.3	784	336.0	72.86

The SAC model yields better results (higher mean, lower variance) compared to PPO with such low timesteps, however it comes at a great expense of time.

I ran some simulations increasing the target training episodes (to $100k$) and keeping the same default learning rate ($\lambda = 0.003$). Although the SAC yields sightly better results after being trained for 100k instead of 25k episodes, these aren't enough to justify the extra time spent on training it. On the other hand, PPO seems to improve greatly while still keeping reasonable times, eventually surpassing SAC, as clearly shown in 4 after around $50k$ timesteps.

```
sac, default parameters
Test reward (avg +/- std): (430.85 +/- 47.79380189940951)

sac, default parameters, 100k
Test reward (avg +/- std): (434.28 +/- 52.99303350441452)

ppo, default parameters
Test reward (avg +/- std): (167.89 +/- 64.66233756987138)

ppo, default parameters, 100k
Test reward (avg +/- std): (290.37 +/- 87.8205733299436)
```

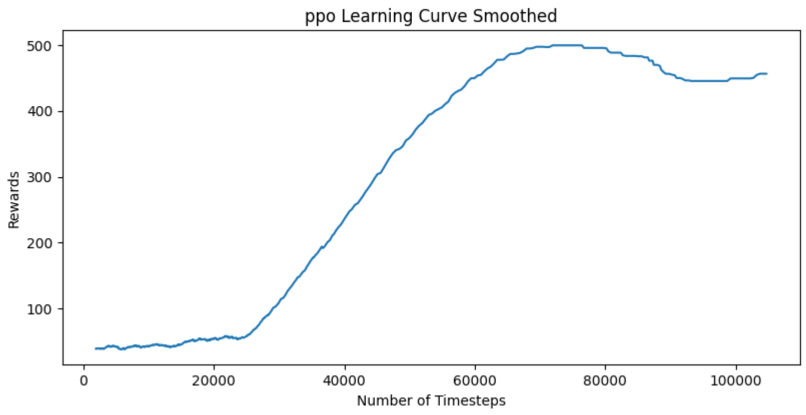

Figure 4: PPO on a 100k target timestep

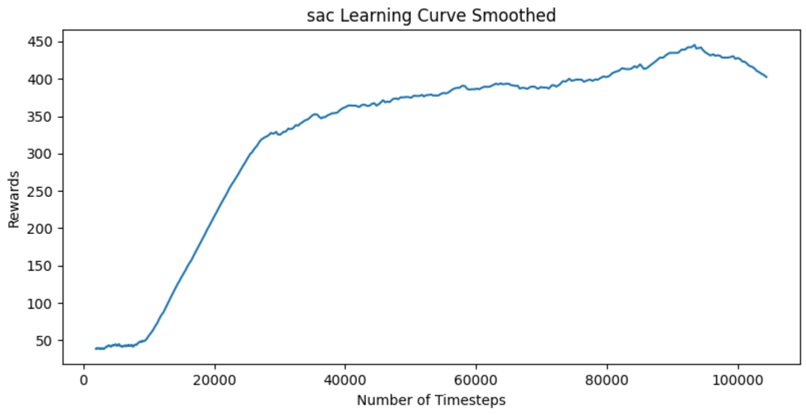

Figure 5: SAC on a 100k target timestep

An important difference between the Actor-Critic methods and REINFORCE, is in the use of one-step returns in one-step actor-critic methods instead of the full return employed by REINFORCE. This decreases the overall variance, computational congeniality, but introduces bias. Although Actor-Critic methods are more complex, mostly due to the computation of the value function used to learn the policy parameter, they are preferred for their better stability and sample efficiency.

References

1. "Reinforcement Learning: An introduction (Second Edition)" by Richard S. Sutton and Andrew G. Barto http://incompleteideas.net/book/RLbook2020.pdf