Exercise 5

Breno Aberle - 876438
ELEC-E8125 - Reinforcement Learning

October 26, 2020

1 Task 1

In this task we are working with a constant variance of 5. However, according to the statements of the TAs in slack we should set sigma = 5 instead of sqrt(5). Due to the variance the plots vary a lot and after every rerun the plots look differently.

1.1 Task 1 a)

Basic REINFORCE without baseline.

In this sub-task we have no baseline and don't calculate discounted rewards. The following code snippet is crucial for this task and figure 1 shows the performance plot:

```
weighted_probs = action_probs * discounted_r
```

1.2 Task 1 b)

REINFORCE with a constant baseline b = 20.

In this sub task we use a baseline. The baseline gets subtracted from the rewards in the following way:

```
weighted_probs = action_probs * (discounted_r - self.baseline)
    # REINFORCE with baseline # T1 b)
```

The performance plots with baseline are illustrated in figure 2.

1.3 Task 1 c)

REINFORCE with discounted rewards normalized to zero mean and unit variance.

We normalize the discounted rewards by subtracting the rewards with the mean and dividing them with the standard deviation of the rewards array and result are shown in figure 3:

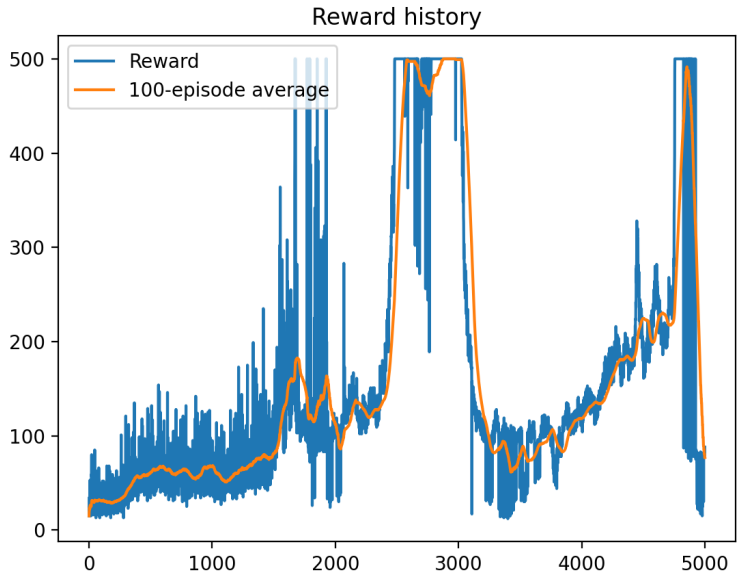

Figure 1: Performance plot with REINFORCE without baseline.

```
discounted_r = discount_rewards(rewards, self.gamma)
discounted_r -= torch.mean(discounted_r)  # for Task 1 c)
discounted_r /= torch.std(discounted_r)
```

2 Question 1.1

How would you choose a good value for the baseline?

A good baseline would be to use the state-value current state. A natural baseline is the state value function, the value of being in that state V(s). Usually these functions are learned by parametric estimators (linear function approximators, neural networks, etc.) [1].

3 Question 1.2

How does the baseline affect the training, and why?

Subtracting a baseline from that return helps reducing the variance and stabilizing the algorithm [1]. Moreover, reducing the variance results in faster convergence and thus speeds up the learning process. The state value V(s) is the expected return starting from state s at time t. The resulting quantity, vt - V(s), removes the dependency on all future states from vt. The quantity vt - V(s), being only dependent on the current state, has lower variance than the original quantity. vt depends on all future states. Intuitively, a baseline function

[1]Source: RL Course by David Silver - Lecture 7: Policy Gradient Methods

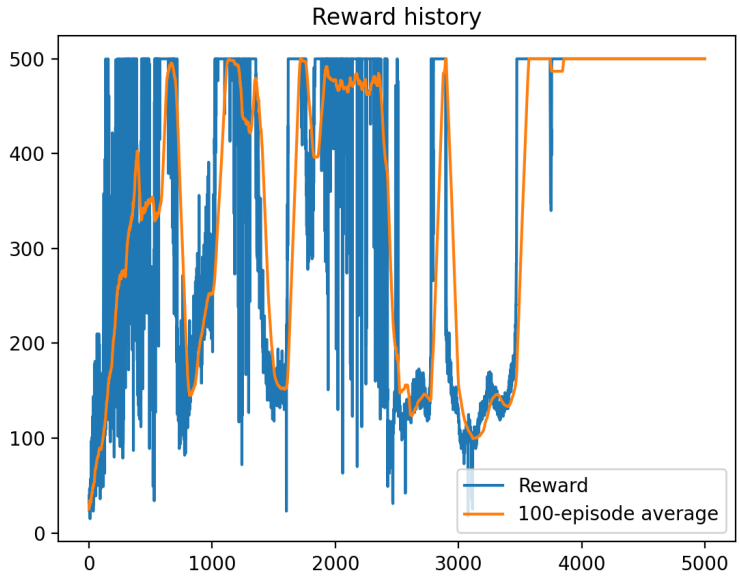

Figure 2: Performance plot with REINFORCE with a constant baseline b = 20.

needs to somehow summarize future return to be able to reduce variance. An arbitrary baseline function may not do that, even when it is unbiased.

4 Task 2

After a big discussion on slack the TAs told us to use sigma = 10 instead of sigma = sqrt(10). The results turn out to be better that way.

a)

The following code snippet is relevant for the exponentially decaying variance and figure 4 shows the performance plot.

```
1 sigma = self.sigma * np.exp((-1) * 5e-4 * episode_number)
```

b)

In this sub task the variance is learned as a parameter of the network. I implemented it as following, so that learned variance is updated automatically by the optimizer. The implementation makes your learned variance automatically updated by the optimizer [2]. The result is shown in figure 5.

```
1 self.sigma = torch.nn.Parameter(torch.tensor([10.]))
```

[2]Source: PyTorch docs: torch.nn.parameter.Parameter

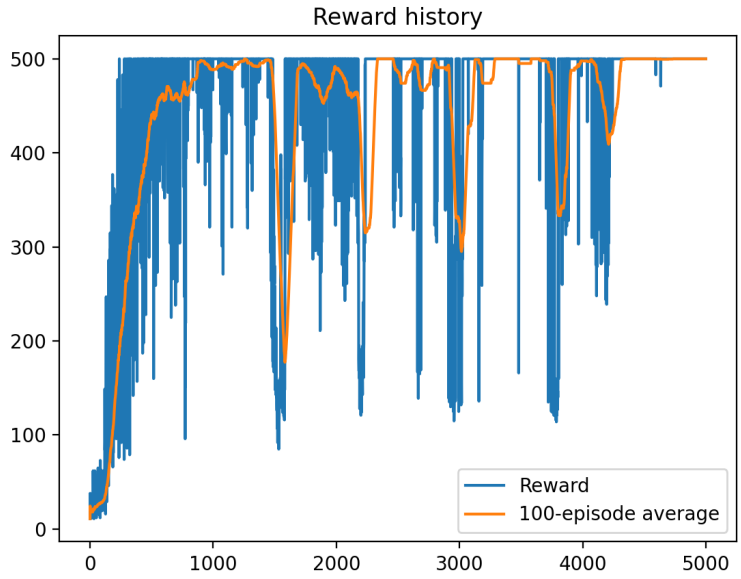

Figure 3: Performance plot with REINFORCE with discounted rewards normalized to zero mean and unit variance.

5 Question 3.1

Compare using a constant variance, as in Task 1, to using exponentially decaying variance and to learning variance during training. Please explain what the strong and weak sides of each of those approaches are.

For the algorithm with constant variance, the one with baseline performed better than the one without. However, the plots are very inconsistent due to high variance. Normalized discounted rewards provided the best result for constant sigma. The advantages are definitely faster computation with price of inconsistent results. Furthermore, the performance strongly depends on the initialization value of the sigma. constant variance:

By using exponentially decaying variance we get better results than with a constant variance. Results are getting more stable regarding variance in performance plots. However, there is still some considerable instability. The decay has higher variance at the beginning which gets smaller over time. This allows more exploration at the beginning and towards the end we have mainly exploitation. It is not so sensitive to the initialization since the value gets smaller over time.

With learning variance during training the best and most stable results were achieved. Variance is a parameter which is included in the optimization process. Therefore, better adjustment and better results are achieved. Variance changes in each iteration slightly defined by the learning rate. Hence, it is a little bit slower to reach convergence. But as soon as it is reached we get stable high rewards with less variance than the before mentioned variants. It is not so dependent on initial value because it gets updated during each parameter update after every iteration. In general, more computation heavy but better results.

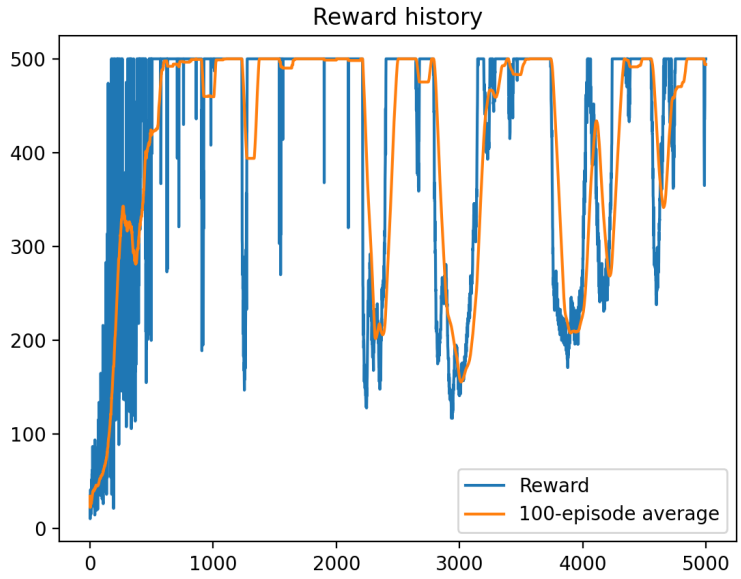

Figure 4: Performance plot with exponentially deaying variance.

6 Question 3.2

In case of learned variance, what's the impact of initialization on the training performance?

The initialization of constant variance has a big impact. Too low variance doesn't allow much exploration and there it is very likely that a not so well policy is found. Higher value results in faster convergence of finding a good policy but also high variance in rewards during exploitation phase. In the case of exponentially decaying variance it beneficial to start with high variance to explore more. Then it is more likely to learn a better policy and after a considerable amount of time steps the policy can be exploited with low variance to increase maximal reward.

7 Question 4.1

Could the method implemented in this exercise be directly used with experience replay? Why/why not?

Like introduced in chapter 4.1 in the course book [2] the REINFORCE algorithm is a on-policy algorithm. In off-policy algorithms the policy gets updated and improved, therefore it changes over time. With experience replay we update the policy based on stored transitions from previous episode. Therefore the policy changes through updates. Consequently, we can't directly apply experience replay on on-policy algorithms. In experience replay we are evaluating actions from a policy that is no longer the current one, since it evolved in time, thus making it no longer on-policy. With importance sampling this problem could be kind of bypassed [3]. Importance sampling is a technique to estimate expectation of function under

[3]Source: Medium article on RL policy gradients and importance sampling

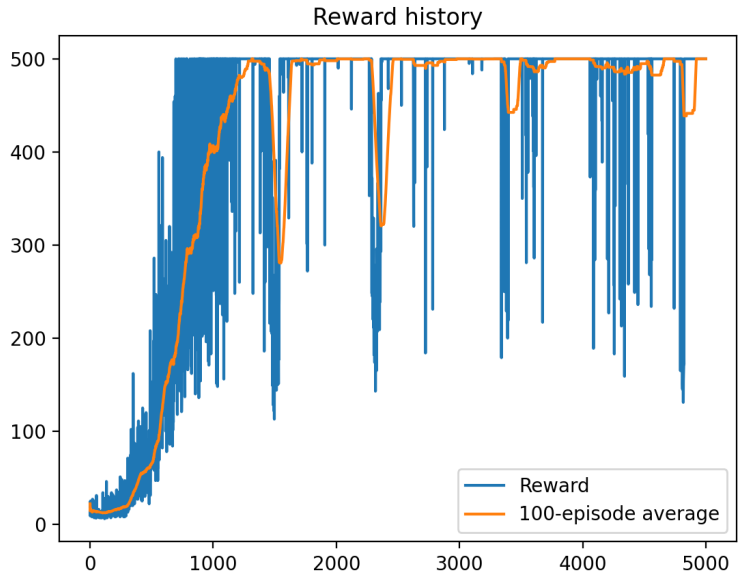

Figure 5: Performance plot with variance learned as a parameter of the network.

distribution p(x) with samples drawn from another distribution q(x). In policy gradient, this technique makes the agent to use off-policy samples drawn from old policy to update current policy. Thus, importance sampled policy gradient can reuses previous samples for training and ensures faster convergence.

8 Question 4.2

Which steps of the algorithm would be problematic to perform with experience replay, if any? Explain your answer.

The questions 4.1 and 4.2 are combined and therefore the answers to 4.2 are already mentioned above.

9 Question 5.1

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?

The agent tends to stay alive indefinitely since it can continuously increase the return by just surviving. However, in real life you can't go on for ever. By physical laws, you always have abrasion and at one point you run out of fuel. Moreover, something can break, the motor stops working, or a material damage occurs due to taken action.

10 Question 5.2

How could the problems appearing in Question 5.1 be mitigated without putting a hard limit on the actions? Explain your answer.

We can mitigate the problems by giving higher reward for balancing with least to no effort. As a consequence, there is less abrasion, because we are moving the least. We should use the least resources by rewarding for less velocity so less fuel gets used. In general, some actions tend to result in breaking something or abrasion and these actions need to get penalized.

11 Question 6

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.

In the paper [3] the problem of discrete action space with policy gradient is addressed. In real-world applications discrete action space occur very frequently. Additionally, the complexity grows exponentially with the action-space dimension. It is challenging to apply existing on-policy gradient based deep RL algorithms efficiently. To effectively operate in multidimensional discrete action spaces, we need to construct a critic to estimate action-value functions, apply it on correlated actions, and combine these critic estimated action values to control the variance of gradient estimations.

References

[1] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Adv. Neural Inf. Process. Syst*, vol. 12, 2000.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Y. Yue, Y. Tang, M. Yin, and M. Zhou, "Discrete action on-policy learning with action-value critic," 2020.