# Exercise 5

Eugeniu Vezeteu - 886240
ELEC-E8125 - Reinforcement Learning

October 17, 2020

## 1 Task 1 - Policy gradient with and without baseline

For this exercise constant variance $\sigma^2 = 5$, was used and defined via:

```
self.sigma = torch.tensor([5.0])
```

in the *Policy* class.
The actions distribution was computed using $\pi_\theta(a_t|s_t) \sim N(f_\theta(s_t), \sigma^2)$, [1].
where $N$ is Gaussian distribution with mean = $f_\theta(s_t)$ (output of the network) and given variance.

In the implementation *torch.distributions.Normal* function was used.

```
action_dist = Normal(action_mean, torch.sqrt(sigma))
```

In the *Agent* class the $get\_action(self, observation, evaluation = False)$ function was implemented.

We take the actions probability by forwarding the current state through our policy, and on the training phase we sample one action, otherwise get the mean.

```
...
actions_probs = self.policy.forward(x)

if evaluation:
    action = actions_probs.mean
else:
    action = actions_probs.sample((1,))[0]

act_log_prob = actions_probs.log_prob(action)

return action, act_log_prob
```

1. *basic REINFORCE without baseline:*
   In basic REINFORCE without baseline method we used the pseudocode from [1] page 328.
   compute the discounted reward $G = \sum_{k=t+1}^{T} \gamma^{k-t-1} * R_k$, in the code the provided *discount_rewards* function was used in the following way:

```
G = discount_rewards(r=rewards, gamma=self.gamma)
```

   $\theta = \theta + \alpha * \gamma^t * G * \nabla ln\pi(A_t|S_t, \theta)$
   The loss was computed

```
loss = torch.sum(-G * action_probs)
```

   Gradient computation and parameter optimization were performed with

```
loss.backward()
.
.
.
self.optimizer.step()
self.optimizer.zero_grad()
```

   The result of basic REINFORCE training is presented in Figure 1.

2. *REINFORCE with a constant baseline b = 20*: For this case a constant baseline b = 20, was added.
   $\theta_{t+1} = \theta_t + \alpha * (G_t - b(S_t) * \nabla ln\pi(A_t|S_t, \theta)$, [1] eq 13.11
   The change in the code is shown in the next code snippet:

```
loss = torch.sum(-(G-self.baseline)*action_probs)
```

   The result of basic REINFORCE training is presented in Figure 2.

3. *REINFORCE with discounted rewards normalized to zero mean and unit variance*:
   On this method the normalization of the discounted rewards was used

```
G = discount_rewards(r=rewards, gamma=self.gamma)
#normalized rewards
G = (G - torch.mean(G)) / torch.std(G)
```

   The result of REINFORCE training with normalized discounted rewards is presented in Figure 3.

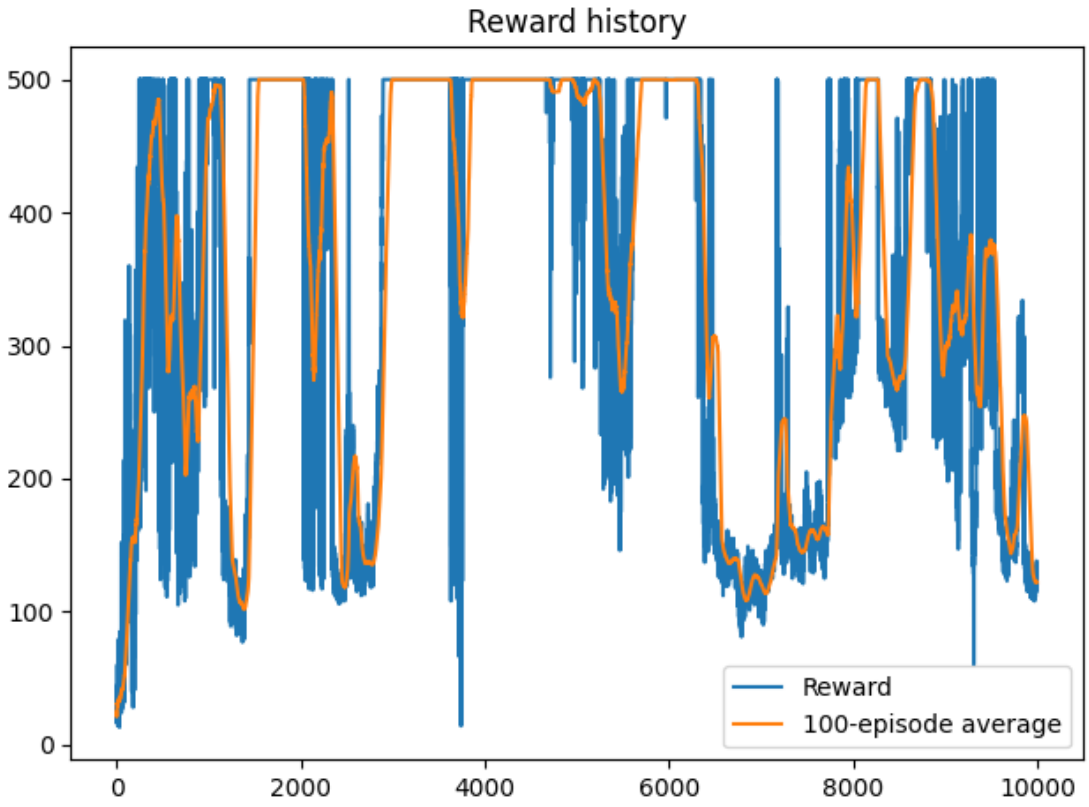


Figure 1: Training basic REINFORCE for 10000 episodes

## 2 Question 1.1 - How would you choose a good value for the baseline?

We can compute the optimal baseline, using the formula given in the lecture slides, [2], slide 20.

$$b_h = \frac{E_\tau[(\sum_{t=0}^{H} \nabla_\theta log\pi_\theta(a_t|s_t))^2 * R_\tau}{E_\tau[(\sum_{t=0}^{H} \nabla_\theta log\pi_\theta(a_t|s_t))]^2}$$

Approximate $b_h$ by empirical mean.
Also, as stated in [1], pg 329, "In some states all actions have high values and we need a high baseline to differentiate the higher valued actions from the less highly valued ones; in other states all actions will have low values and a low baseline is appropriate.", so to choose a good baseline we need to set it as an estimation of the state value $v(s_t, \theta)$.

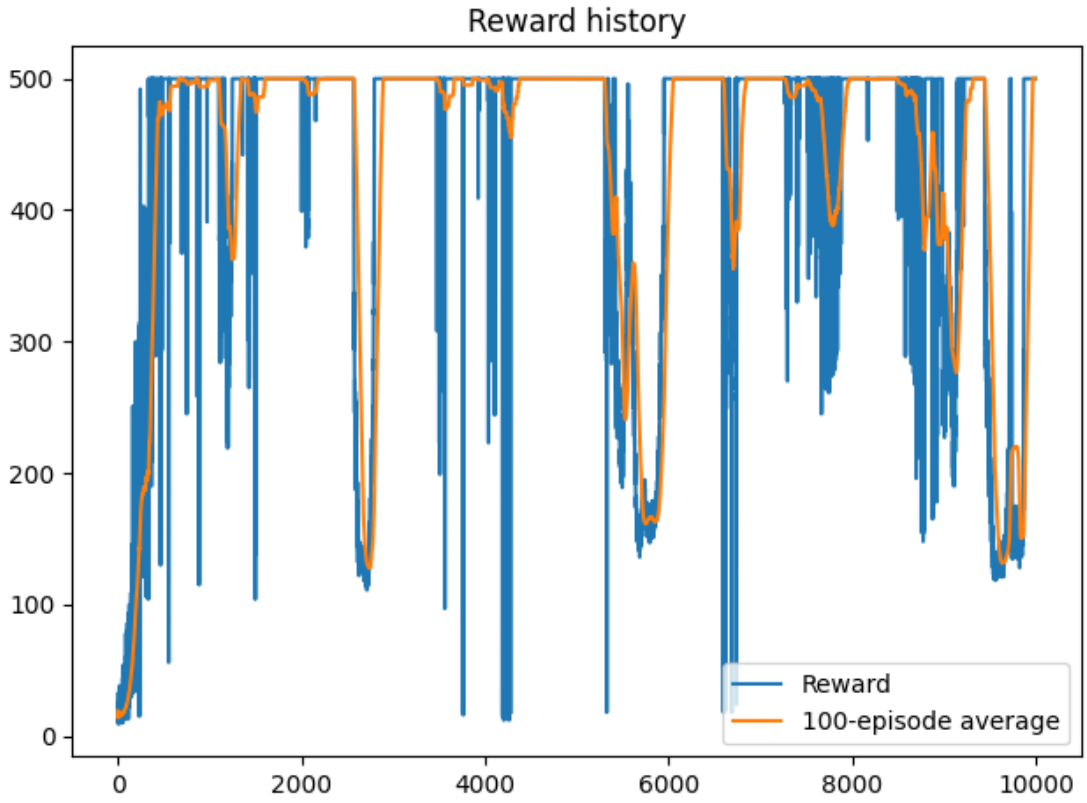


Figure 2: Training REINFORCE with constant baseline = 20 for 10000 episodes

# 3 Question 1.2 - How does the baseline affect the training, and why?

The baseline speed up the training, by decreasing variance, because it reduces the variance of the gradient estimation.

$$\nabla_\theta R(\theta) = E_\tau[\nabla_\theta log p_\theta(\tau)(R(\tau) - b)] = E_\tau[\nabla_\theta log p_\theta(\tau) R(\tau)]$$

, [2], slide 19.
We can notice the difference in the Figures 1 and 2, that the introduction of the baseline, has reduced the instability of the agent.

# 4 Task 2 - Choosing the value of variance

On this task, two cases of adjusting the variance were used. In both cases the value of $\sigma_0^2 = 10$

1. exponentially decaying variance: - $\sigma^2 = \sigma_0^2 * e^{-c*k}$, were c $= 5 * 10^{-4}$ and k is the number of episode.

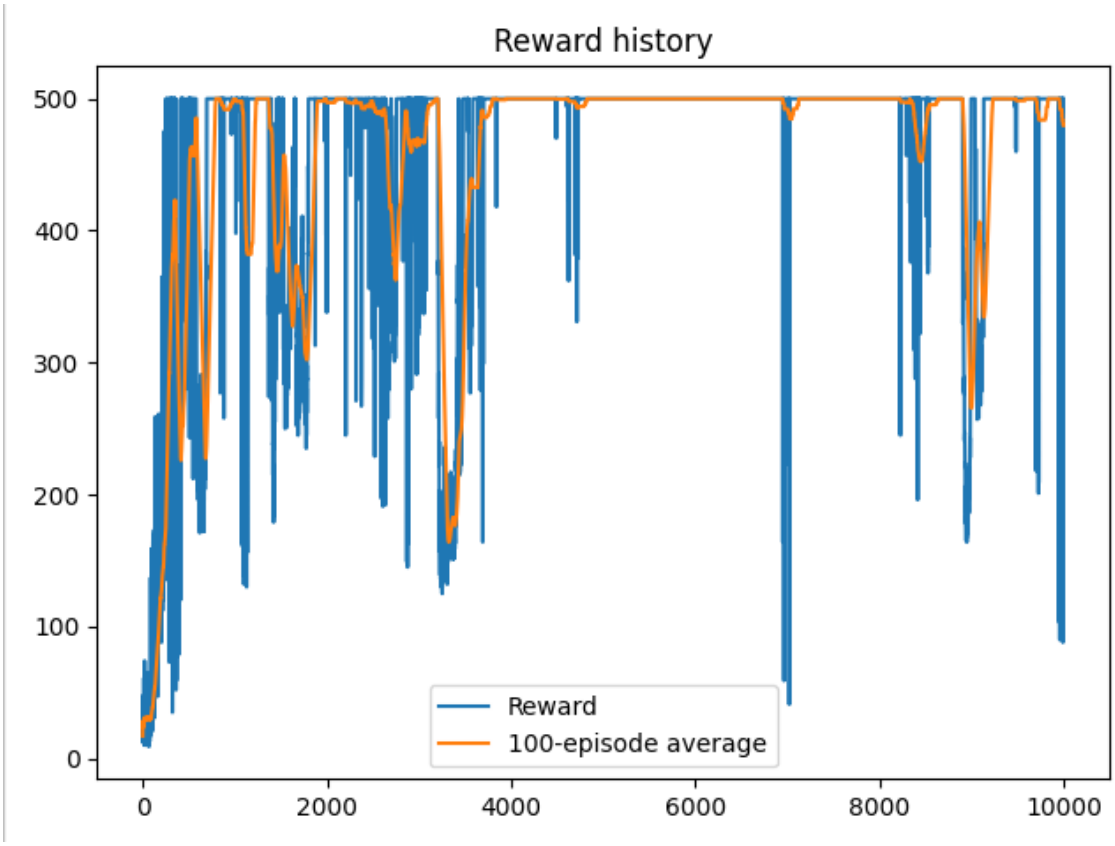


Figure 3: Training REINFORCE with normalized discounted rewards for 10000 episodes

```
c = 5e-4 #T2
self.variance = self.policy.sigma * np.exp(-c *
    episode_number)
```

The training result is presented in Figure 4.

2. variance learned as a parameter of the network: - for this task the sigma was declared as:

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

to be automatically updated by the optimizer.

The training result is presented in Figure 5.

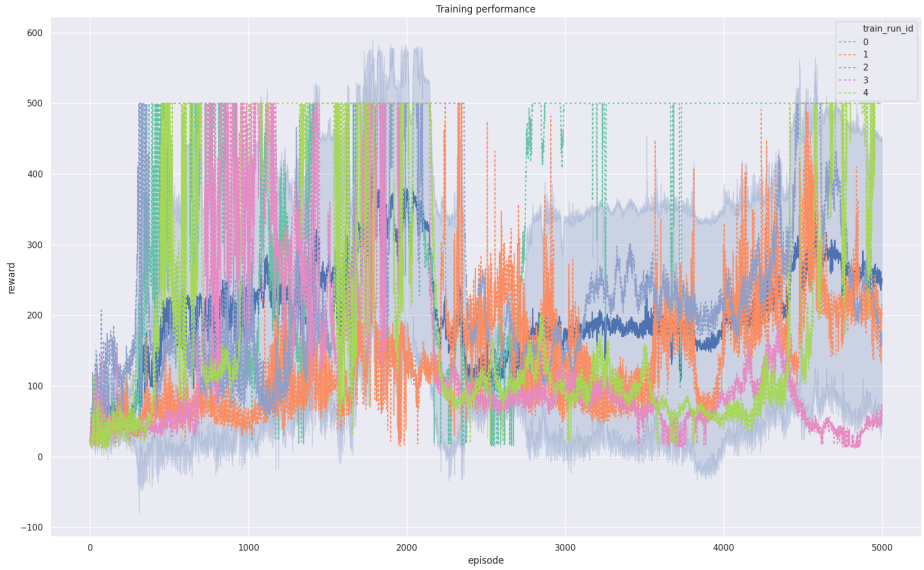


Figure 4: Training REINFORCE with normalized discounted rewards for 5000 episodes with exponentially decaying variance

# 5 Question 3.1 - strong and weak sides of exponentially decaying variance and to learning variance during training

1. **constant variance:** (*weak*) - High dependence on the initial value, is unstable during learning. (*strong*) - Lower computational complexity and fast convergence (if we set a big init value, it will explore faster).

2. **learning variance**: (*strong*) - Lower dependence on the initial value, better stability and performance, due to the fact that the variance is adapted/learned (best approach). (*weak*) - Higher computational complexity (parameters must be updated by the optimizer).

3. **exponentially decaying variance**: (*weak*) - Still dependent on initial value, exponential decay parameter $c$ must be selected, higher computational complexity (in comparison with constant value). (*strong*) - Higher exploration in the beginning and lower in the end (increased exploitation in the end), better stability.

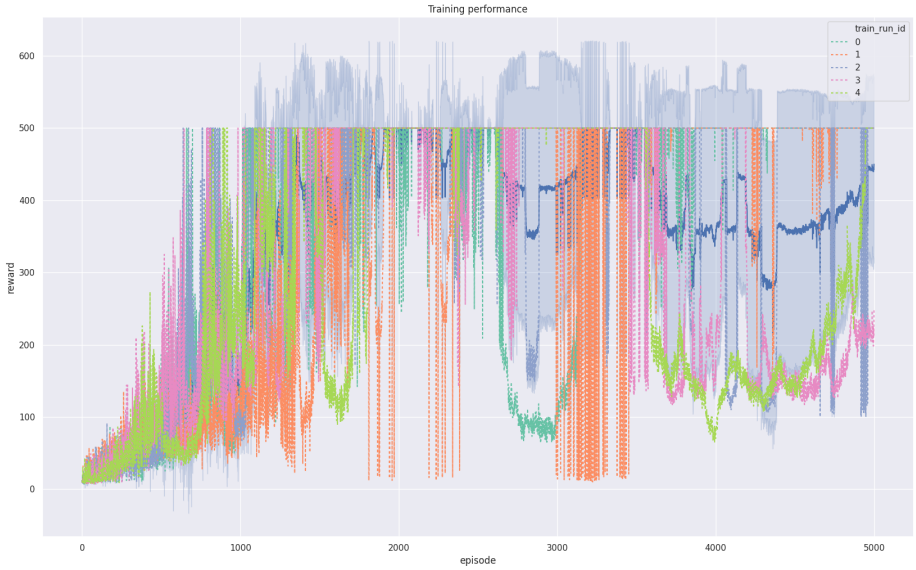


Figure 5: Training REINFORCE with normalized discounted rewards for 5000 episodes with learned variance

# 6 Question 3.2 - In case of learned variance, what's the impact of initialization on the training performance?

The initialization of the learned variance controls the exploration and exploitation of the policy. Too small variance results in a smaller exploration and higher exploitation, which is bad at the beginning of the training. We want our policy to explore at the beginning of the training. A higher variance ensures a better exploration at the beginning of the training, while being adapted/decreased during the learning.

# 7 Question 4.1 - Could the method implemented in this exercise be directly used with experience replay?

The REINFORCE algorithm is on-policy method, while Experience Replay is off-policy method, because it learns from stored samples, that weren't generated by the current policy. But we can adapt REINFORCE algorithm to be use it with Experience Replay, the trick that allows policy gradient to be used with off-policy transitions is the importance sampling. [2] slide 23 (Off-policy policy gradient).

# 8 Question 5.1 - What could go wrong when a model with an unbounded continuous action space

In a real-world scenario it is not recommended to use policy gradient with unbounded continuous action space, because the action predicted by the agent may be too small or too high (does not fall within the limits of the system in which it is used), it may damage the system. Let's assume that we want to control the electric motor and the actions are the current sent to the motor, the actions must be adapted to the limits of the system, because we have already noticed that the policy gradient is unstable, and an unbounded action can burn down the system.

# 9 Question 5.2 - How could the problems appearing in Question 5.1 be mitigated without putting a hard limit on the actions?

Without the hard limit of the actions, we can still avoid predicting an unbounded action. REINFORCE method is unstable because of too high variance, we can use TD approaches, which has smaller variance (but cannot handle stochastic policies or continuous action spaces [2] lecture 6, slide 4). We have already seen how the baseline is able to reduce the variance of REINFORCE method. We shall use ACTOR-CRITIC methods, combine both PG and TD, where value function (Critic) will restrict the variance of the PG (actor)

# 10 Question 6 - Can policy gradient methods be used with discrete action spaces?

Yes, the policy gradient algorithm can be used with discrete action spaces. If the action spaces is not too high, its even simpler than in continuous action space case, we can use the softmax function in the last layer, and treat it as a classification problem, where the action with the highest probability will be selected. However, the performance will decrease as the number of discrete action starts to increase.

# References

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

[2] V. Kyrki, *Lecture slides - Lecture 5, slide 19*. Aalto University, 2020.