Robot Learning - Laboratory 3
REINFORCE, PPO, SAC

Andrea Delli (S331998)
Robot Learning course (01HFNOV)
Master's Degree in Computer Engineering
Artificial Intelligence and Data Analytics
Politecnico di Torino

Abstract: This laboratory explores three popular reinforcement learning algorithms: REINFORCE, Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC), applied to the continuous CartPole environment. REINFORCE, a policy gradient method, is simple but suffers from high variance in its updates, making it less effective for complex environments. PPO improves on this by introducing clipped objectives to stabilize training, achieving more reliable learning. SAC further enhances performance by incorporating entropy regularization, encouraging exploration while maintaining efficient learning. Experiments demonstrate that SAC and PPO outperform REINFORCE in terms of stability and convergence, and highlight the differences in training the SAC and PPO methods.

1 Introduction

The Cart-Pole environment is a fundamental and widely-used benchmark in reinforcement learning (RL), designed to test control strategies and decision-making algorithms. The setup consists of a cart that can move along a one-dimensional track and a pole mounted on the cart, free to pivot around its attachment point. The primary goal of the task is to balance the pole upright by applying forces to the cart to move it left or right.

Key components of the environment include a state observation vector s containing four elements: the cart's position x and velocity $\dot{x}$, the pole's angle θ relative to the vertical axis, and the pole's angular velocity $\dot{\theta}$. The agent's objective is to maintain the pole in a vertical position while keeping the cart within the track's boundaries. Any significant deviation, such as the pole falling or the cart moving out of bounds, results in a failure.

$$s = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

1.1 REINFORCE

In this laboratory we focus on the **REINFORCE** algorithm, a policy-based, model-free reinforcement learning method used to optimize a parameterized policy $\pi_\theta(a|s)$ by maximizing the expected cumulative reward. The goal is to directly adjust the policy parameters θ to improve performance. REINFORCE uses the policy gradient theorem to estimate gradients without requiring knowledge of the environment's dynamics:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right]$$

where:

- G_t is the cumulative reward (return) from time step t onward

- $\pi_\theta(a_t|s_t)$ is the policy's probability of taking action a_t in state s_t at time t
- $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the policy gradient

The policy parameters are updated iteratively using **gradient ascent**:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\theta_t)$$

where α is the learning rate.

The gradient tries to increase the probability of paths with positive reward R and decrease the probability of paths with negative reward R. The problem is that learning is slow and has high variance.

To stabilize learning, we can use a **baseline** $b(s)$, which is subtracted from the cumulative reward:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t)) \right]$$

The update rule becomes:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(G_t - b(b_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

1.2 Actor-Critic approaches

In this laboratory, we will also explore two popular **actor-critic** methods available in the Stable-Baselines 3 library: **Proximal Policy Optimization (PPO)** and **Soft Actor-Critic (SAC)**. Both algorithms leverage the actor-critic framework, where a policy (actor) is optimized using feedback from a value function (critic), combining the strengths of policy-based and value-based methods.

1.2.1 Proximal Policy Optimization (PPO)

PPO is a policy-gradient method that introduces a clipped surrogate objective to ensure stable and reliable updates. It optimizes the policy $\pi_\theta(a|s)$ by maximizing a clipped objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E} \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t \right) \right]$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is the probability ratio of the new and old policies
- A_t is the advantage estimate
- ϵ is a clipping parameter to limit changes to the policy

PPO ensures balanced exploration and exploitation by constraining updates, making it robust and widely applicable.

1.2.2 Soft Actor-Critic (SAC)

SAC is an off-policy algorithm that optimizes a stochastic policy while encouraging exploration through an entropy-regularized objective. The goal is to maximize both the expected return and the policy's entropy, promoting diverse actions. The SAC objective is:

$$J_\pi = \mathbb{E} \left[Q(s, a) - \alpha \log \pi(a|s) \right]$$

where:

- $Q(s, a)$ is the action-value function
- $\log \pi(a|s)$ is the policy's entropy
- α is a temperature parameter controlling the entropy weight

SAC uses separate networks for the policy (actor) and value functions (critic), updating them iteratively. Its entropy term makes it particularly effective in continuous action spaces.

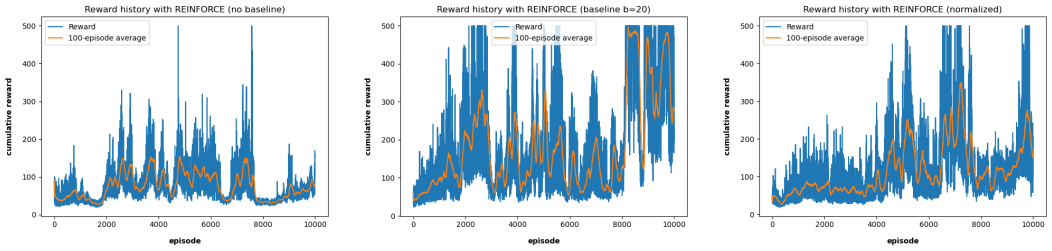
2 Experiments on REINFORCE

The trained model outputs a probability distribution over actions that follow a Normal distribution, with mean μ given by the model output and fixed standard deviation $\sigma^2 = 5$.

REINFORCE has been tested using three different settings:

1. REINFORCE without baseline
2. REINFORCE with constant baseline $b = 20$
3. REINFORCE with discounted rewards normalized to zero mean and unit variance

The results of the training are shown in the reward history plots reported in Fig. 1, where the training has been performed over 10000 episodes. The first plot (Fig. 1a) shows that without any baseline the model performs poorly, has a high reward variance and doesn't show any clear improvement in the average reward history. The second plot (Fig. 1b) shows a better performance, showing some peaks that reach the optimal value of 500, even though the reward variance is still high and learning is clearly unstable. The last plot (Fig. 1c) also shows a good model performance, with some episodes reaching the optimal result, and a more steady learning with less reward variance.



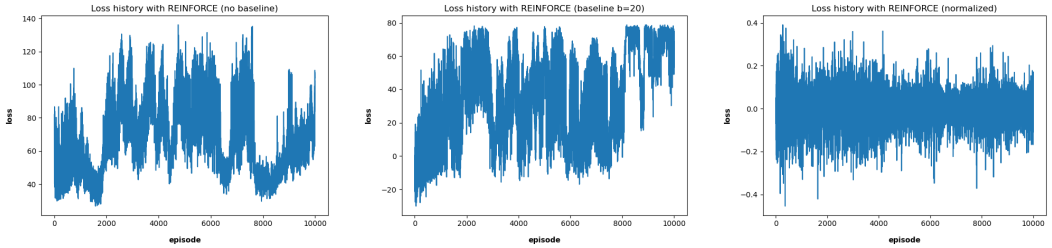
(a) Reward history without baseline

(b) Reward history with constant baseline $b = 20$

(c) Reward history with discounted rewards normalized to zero mean and unit variance

Figure 1: Reward history for REINFORCE with different baselines and discounted rewards

The plots reported in Fig. 2 show the comparison between the loss during training for the three implemented versions of REINFORCE. Since we're using gradient ascent, a higher value of loss is considered better, thus we'd want the value of the loss to be increasing during training. The only clear improvement, as expected from the reward history plots (Fig. 1), is given by the plot in Fig. 2b with constant baseline, in which we have a slight increase in the loss during training. The other noticeable difference is that for the normalized discounted rewards (shown in Fig. 2c) the loss values are in the range $[-1; +1]$.



(a) Loss history without baseline

(b) Loss history with constant baseline $b = 20$

(c) Loss history with discounted rewards normalized to zero mean and unit variance

Figure 2: Loss history for REINFORCE with different baselines and discounted rewards

2.1 Considerations on REINFORCE baselines

A good baseline value in the REINFORCE algorithm should reduce variance in the gradient estimates while maintaining the unbiased nature of the updates. The baseline can be any function of the state, as long as it does not depend on the actions.

The state value function, $V(s)$, is often a good choice for the baseline because it represents the expected return from a state under the current policy. This makes it effective at normalizing the advantage, $A(s, a) = G_t - b(s)$, by centering the returns around zero, which stabilizes learning. Alternatively, using a simple constant baseline (as we did with $b = 20$) can still significantly reduce variance even if computational resources are limited.

The baseline affects training by reducing the variance of the policy gradient estimates. Without a baseline, updates depend directly on the raw returns, which can vary widely, especially in stochastic environments. This high variance can slow down learning and make optimization unstable.

By subtracting a baseline, the advantage term ($A(s, a) = G_t - b(s)$) is centered, focusing the updates on the relative improvement of actions with respect to the baseline. This reduces the noise in gradient estimates, allowing the algorithm to make more consistent progress. However, the baseline must be independent of the actions to ensure unbiased gradient updates. Improperly chosen baselines can lead to biased updates, negatively affecting the policy's ability to converge to the optimal solution.

The average performance of the generated models during testing is reported in Table 1, where the performance is measured as the average number of timesteps for which the model survives (1 timestep $\implies$ $+1$ reward). The tested models are obtained by selecting only the models performing better during training, in order to avoid saving only the last network states which may be sub-optimal. By analyzing these results, we can conclude that a baseline significantly improves the algorithm convergence speed and learning stability, increasing the efficiency of REINFORCE.

Model	Average reward
REINFORCE (no baseline)	192.17
REINFORCE (baseline b=20)	499.81
REINFORCE (normalized)	275.15

Table 1: Performance comparison between different REINFORCE implementations, averaged over 100 test episodes

2.2 Real-world problems

In a real physical system, an unbounded continuous action space can lead to unsafe or unfeasible actions. The model may produce excessively large force or torque values, which could damage the physical system or its environment. Continuous actions might result in oscillations or instability, as the system may overcompensate in an attempt to maximize the reward. Unbounded actions can lead to unnecessarily high energy consumption, making the system impractical in real-world scenarios.

The problems can be mitigated by applying techniques that regularize or limit the action space, for example we could normalize the action outputs of the model by scaling them to a range suitable for the physical system (e.g. by applying a $\tanh(\cdot)$ activation function at the output layer). Another approach is to define the reward functions adding a penalty for actions that deviate significantly from the desired range or for actions that consume excessive energy. This incentivizes the policy to produce reasonable actions. These approaches maintain flexibility in the action space while aligning it with the constraints of a real physical system.

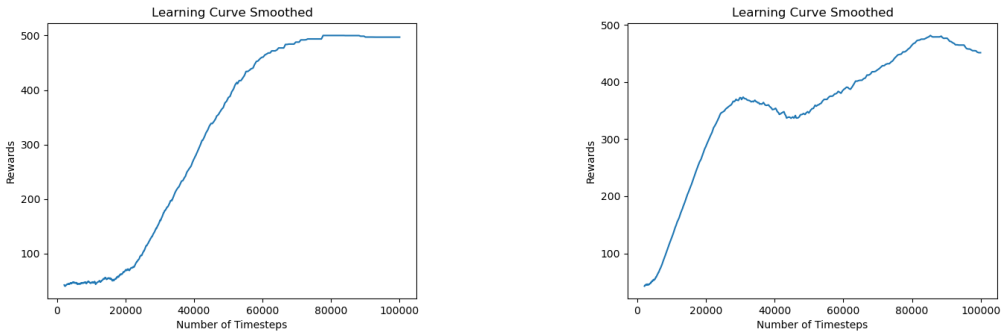
Policy gradient methods can be used with **discrete action spaces**, for example by using as output a probability distribution over the available actions, often using a softmax function. The action to take is then sampled from this distribution. The algorithm adjusts the probabilities of actions to maximize the expected reward, as seen in the previous laboratory about tabular Q-Learning.

3 Experiments on PPO and SAC

This section presents the results of the trained Actor-Critic models implemented using the Stable-Baselines3 library [1]. The plots in Fig. 3 show the learning curves (average reward history) during training of the two models, both trained over 100000 episodes.

In the first plot (Fig. 3a), the PPO agent demonstrates a steady and smooth progression in rewards over time, eventually stabilizing around the optimal reward level of 500 after approximately 60000 timesteps. This behavior highlights the robustness and efficiency of PPO, which is known for its relatively fast convergence.

In contrast, the second plot (Fig. 3b) shows the learning curve for SAC. While the rewards also increase over time, the curve is less smooth compared to PPO, with minor fluctuations even after the rewards stabilize around the same optimal level of 500. SAC's convergence occurs more slowly, reflecting the algorithm's tendency to require more time and computational resources during training.



(a) PPO reward history

(b) SAC reward history

Figure 3: Loss history for REINFORCE with different baselines and discounted rewards

A comparison between the two models can be seen in Table 2. While both algorithms reach an optimal final performance level, PPO exhibits faster and more stable learning, whereas SAC has a much longer training time and shows slightly more variability during the learning process and in the resulting performance.

Model	Average reward	Reward std deviation	Training episodes	Training time [s]
PPO	500	0	100000	50.98
SAC	499.24	5.34	100000	1958.86

Table 2: Performance comparison between different PPO and SAC, averaged over 100 test episodes

The used metrics and hyperparameters used to train the two final models are reported in Table 3 for PPO and Table 4 for SAC.

Measure	Value
Learning rate	0.0003
Number of steps per update	2048
Batch size	64
Number of epochs	10
GAE λ	0.95
Clip range	0.2
Total timesteps	100000

Table 3: Metrics and hyperparameters used for training the PPO model

Measure	Value
Learning rate	0.0003
Batch size	64
Total timesteps	100000
Buffer size	1000000
Learning starts	100
τ	0.005
γ	0.99
Target update interval	1

Table 4: Metrics and hyperparameters used for training the SAC model

4 Conclusion

4.1 Comparison between PPO, SAC and REINFORCE

The analysis of the trained policies in terms of reward and time consumption reveals distinct performance characteristics for PPO and SAC. By looking at Table 2 we see that both algorithms achieve similar average rewards, with PPO stabilizing at exactly 500 and SAC slightly lower at 499.24. However, the variability in rewards is slightly different: PPO exhibits perfect stability (std deviation = 0), while SAC has a small fluctuation (std deviation = 5.34). This indicates that PPO produces a more consistent policy compared to SAC.

In terms of training time, there is a huge difference between the two methods. PPO completes training in just 50.98 seconds, while SAC requires a significantly longer time of 1958.86 seconds. This highlights PPO's computational efficiency and faster convergence relative to SAC, which is known for its sample efficiency but also for being computationally expensive.

Comparing these results to the previously obtained performance of the REINFORCE algorithm (see Table 1), several differences emerge. REINFORCE typically has less stability in learning compared to both PPO and SAC. Unlike PPO, which uses a clipped objective function to prevent large policy updates, and SAC, which incorporates entropy regularization for stability, REINFORCE lacks these advanced techniques, often resulting in higher variability and slower convergence.

The observed results can be explained by the differences in the underlying algorithm. PPO benefits from an efficient and robust policy optimization process that balances exploration and exploitation while avoiding instability through gradient clipping. SAC is optimal in high-dimensional continuous action spaces due to its entropy-maximization approach, which encourages exploration.

The performance differences observed for REINFORCE are likely due to its reliance on Monte Carlo updates, which introduce high variance in gradient estimates, leading to slower convergence and less stability. PPO and SAC, on the other hand, mitigate these issues through more advanced update mechanisms.

References

[1] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021. URL http://jmlr.org/papers/v22/20-1364.html.