Exercise 3 - Robot Learning Course

Andrea Ongaro
Computer Engineering
Politecnico di Torino, Italy
s329706@studenti.polito.it

Abstract: This article is the third in a series of three reports for the Reinforcement Learning laboratory course at Politecnico di Torino. This report is about policy gradient reinforcement learning algorithms within the OpenAI Gym environment, focusing on the CartPole problem. This process started implementing the naïve REINFORCE algorithm and then exploring more advanced Actor Critic methods like Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC), also leveraging on the use of well-known libraries such as Stable-Baselines3.

Keywords: Reinforcement Learning, Robots, AI

1 Introduction

1.1 The system - CartPole-v0

CartPole-v0 system from OpenAI is part of the classic control environment. The starting code it's been provided by Andrea Protopapa in the course of Reinforcement Learning. Information about this system is sourced from the official documentation [1] and its GitHub repository. Below is the definition provided by the authors for CartPole-v0

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson in "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem". A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.[1]

For the following procedures, it's been used a modified version of the CartPole environment implemented in cp_cont.py the standard CartPole uses discrete actions (applying force of either -10 or 10), while in this version it takes actions in the form of a single real number, which represents an arbitrary force value.

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

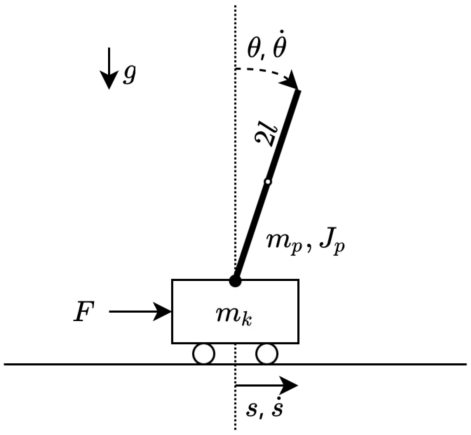

Figure 1: Graphical explation about the whole system

The observation is a four element vector:

$$\boldsymbol{s} = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}$$

Figure 2: Observation vector

where

- x: the position of the cart
- $\dot{x}$: the velocity,
- θ: the angle of the pole
- $\dot{\theta}$: the angular velocity of the pole.,

2 Policy gradient

2.1 REINFORCE algorithm

REINFORCE is a Policy-gradient method, a subclass of Policy-based methods that aims to optimize the policy directly by estimating the weights of the optimal policy using gradient ascent iteratively searching for optimal parameters that maximize the objective function. [PROF]

The REINFORCE algorithm can be applied to a wide range of problems, including control tasks like robotics. However, it has some limitations, such as high variance in the gradient estimates and slow convergence. To address these issues, various extensions and improvements have been proposed, such as using baseline functions to reduce variance, incorporating advantage estimates, and employing trust region methods. [https://advancedoracademy.medium.com/an-introduction-to-the-reinforce-5470a376b3ab]

2.2 Baseline value

2.3 What is?

A baseline is a value subtracted from the returns (or rewards) before calculating the gradient of the policy. Its primary purpose is to reduce the variance of gradient estimates, which can make learning more stable and efficient. Importantly, the baseline does not introduce any bias into the gradient computation.[CHATGPT]

Policy gradient methods update the policy parameters based on the estimated advantage of actions:

$\nabla J(\theta) \propto \mathbb{E}[\nabla \log \pi(a|s;\theta) \cdot A(s,a)]$

where A(s, a) is the advantage function, which measures how much better an action is compared to some baseline.

Subtracting a baseline b from the rewards or returns does not bias the gradient because:

$\mathbb{E}[\nabla \log \pi(a|s;\theta) \cdot b] = 0$

This is because the expectation of the gradient with respect to the policy's probability distribution over actions sums to zero.

In this paper it will be analyzed three different methods, the first one without a baseline. The second with a costant baseline setted to 20, Simple to implement but may not capture the variability of rewards in different states. ¡¡¡¡¡

State-Dependent Baseline: • A baseline that depends on the state, such as the value function V(s) , which estimates the expected return from a state:

$A(s,a) = R\_t - V(s)$

• Used in Actor-Critic methods, where the critic network estimates V(s) .

Learned Baseline: • A separate neural network trained to approximate the baseline (e.g., V(s)). • Dynamically adapts to the agent's performance.

If b is a good approximation of the expected return $\mathbb{E}[G_t]$, the variance of $G_t - b$ is much smaller than G_t

2.4 Why is important

A good baseline value is typically chosen based on the following considerations: Mean of Rewards: • The baseline should approximate the average reward over episodes or across states. This reduces variance without introducing bias. • Empirical Tuning: • Start with simple values (e.g., 0 or the mean reward for a few episodes) and adjust based on training stability. • Learned Baseline: • Use a separate network to estimate the value function (V(s)) for a state, as in Actor-Critic methods. This allows the baseline to adapt dynamically.

2.5 Training

Variance Reduction: • A baseline reduces the variance of gradient estimates by subtracting a constant or state-dependent value from rewards. This makes learning more stable and efficient. • No Bias Introduction: • Subtracting the baseline does not bias the gradient estimation because it does not depend on the actions taken. • Impact of Poor Baseline: • If the baseline is poorly chosen (e.g., far from the expected reward), it may not reduce variance effectively and could slow down training.

3 Actor Critic

In Reinforce, we want to increase the probability of actions in a trajectory proportionally to how high the return is. Given the stochasticity of the environment (random events during an episode) and stochasticity of the policy, trajectories can lead to different returns, which can lead to high variance. Consequently, the same starting state can lead to very different returns. Because of this, the return starting at the same state can vary significantly across episodes.

The Actor-Critic method is a combination of Policy-Based and Value-Based methods: .
there are two function approximations (two neural networks): Actor, a policy function parameterized Critic, a value function parameterized At each timestep, t, we get the current state from the environment and pass it as input through our Actor and Critic.

We can stabilize learning further by using the Advantage function as Critic instead of the Action value function.

The idea is that the Advantage function calculates the relative advantage of an action compared to the others possible at a state:

this function calculates the extra reward we get if we take this action at that state compared to the mean reward we get at that state.

The extra reward is what's beyond the expected value of that state.

If A(s,a) ¿ 0: our gradient is pushed in that direction. If A(s,a) ¡ 0 (our action does worse than the average value of that state), our gradient is pushed in the opposite direction.

4 Continuous Environment

Algorithms designed for continuous action spaces, like PPO and SAC, must handle a continuous range of values for each action dimension.

PPO: Uses a stochastic policy, typically modeled by a Gaussian distribution, to output continuous actions. This allows PPO to handle the fine-grained control needed in continuous action environments effectively. Provides a good balance between exploration and exploitation, making it suitable for environments with continuous state and action spaces. It usually converges faster and is more stable.

SAC: Also employs a stochastic policy with a Gaussian distribution, but it enhances exploration through entropy regularization. The continuous action space allows SAC to leverage the entropy term to maintain diverse action selection, preventing premature convergence. Offers robust exploration through entropy regularization, often leading to better performance in environments with continuous action spaces. However, it may require more computational resources and longer training times.

4.1 Proximal Policy Optimization (POC)

PPO is an on-policy algorithm that balances simplicity and performance. PPO uses a stochastic policy, which means it outputs a probability distribution over actions rather than a single deterministic action. The policy is optimized using a surrogate objective function. Clipping Mechanism: To prevent large updates that could destabilize training, PPO introduces a clipping mechanism. The probability ratio between the new and old policies is clipped to ensure the policy does not change too drastically between updates. Advantages Estimation: PPO relies on advantage estimation, which is the difference between the expected return (value function) and the actual return. This helps the algorithm focus on actions that are better than average. Epochs and Batching: The algorithm updates the policy by iterating over multiple epochs and mini-batches, which ensures efficient and stable learning.

we want to avoid having too large of a policy update.

For two reasons:

We know empirically that smaller policy updates during training are more likely to converge to an optimal solution. A too-big step in a policy update can result in falling "off the cliff" (getting a bad policy) and taking a long time or even having no possibility to recover.

So with PPO, we update the policy conservatively. To do so, we need to measure how much the current policy changed compared to the former one using a ratio calculation between the current and former policy. And we clip this ratio in a range $[1-\theta, 1+\theta]$, $[1-\theta, 1+\theta]$, meaning that we remove the incentive for the current policy to go too far from the old one (hence the proximal policy term).

4.2 Soft Actor Critic (SAC)

SAC is an off-policy algorithm designed for continuous action spaces. Here's how SAC learns:

Actor-Critic Framework: SAC employs an actor (policy) and two critics (value functions). The actor selects actions, while the critics evaluate them. Entropy Regularization: SAC maximizes both the expected return and the entropy of the policy. Entropy regularization encourages exploration by adding a term to the objective function that measures the randomness of the policy. This helps prevent premature convergence to suboptimal policies. Off-Policy Learning: SAC leverages a replay buffer to store and reuse past experiences, which improves sample efficiency. The critics are updated using batches of experiences drawn from the replay buffer. Soft Q-Learning: SAC uses a soft version of the Bellman equation to update the critics. This involves minimizing the difference between the predicted Q-values and the target Q-values, which include the entropy term.

4.3 Real-world control problems

4.4 Risks with Unbounded Continuous Action Spaces and Reward Function

In a physical system: • High Risk of Instability: With unbounded actions, the policy may propose extreme actions, leading to system instability or damage. • Reward Gradient Issue: A survival-based reward like +1 offers no penalty for extreme actions unless explicitly constrained. This can mislead the agent into learning aggressive policies that maximize survival without safety considerations.

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? 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.

4.5 Discrete action spaces

policy gradient methods can be used with discrete action spaces. Here's why: • Action Selection: • Instead of sampling from a continuous distribution (e.g., Normal), the policy outputs a probability distribution over discrete actions (e.g., Categorical distribution). • Log-Probability: • The log-probability of the selected action can still be computed, enabling the gradient update.

Potential Issues with Discrete Action Spaces 1. Sampling: • Actions are sampled from a Categorical distribution. This introduces randomness but is straightforward to implement. 2. Gradient Estimation: • The gradients are computed from the log-probabilities, which are well-defined for discrete actions. 3. Exploration: • Ensuring sufficient exploration is more challenging in discrete spaces but can be addressed with techniques like entropy regularization.

5 Experiments

In this section you'll see different tests applying different methods to solve the same problem, stabilize the pole.

5.1 REINFORCE without baseline

The first method is to use REINFORCE wihout a baseline, and as you can see from the graph is not suitable to solve this task, in fact, the agent does not demonstrate clear convergence to an optimal policy. The reward curve is noisy, indicating high variability in the rewards as explained earlier while The 100-episode average curve shows that learning progresses very slowly, with periods of improvement followed by plateaus (e.g., between episodes 300-700) that indicate that the agent struggles to make consistent progress or declines.

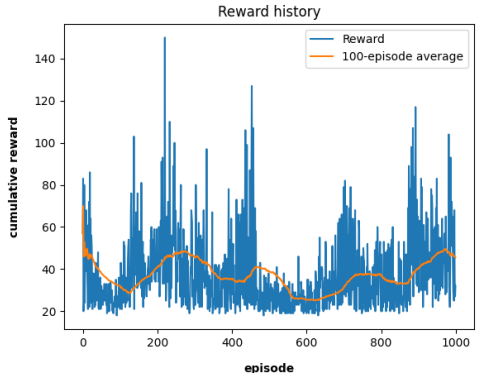

Figure 3: Reward history - Basic REINFORCE no baseline

5.2 REINFORCE with a constant baseline $b = 20$

Compared to the graph without a baseline, the reward curve (Graph 4) is less noisy, indicating more stable learning. This is because the constant baseline reduces the variance of the policy gradient updates. The agent achieves higher cumulative rewards more quickly with a baseline than without. The orange curve indicates steady progress even if sometimes it still has some little plateus (e.g. near the 300 episodes and near 700 episodes), whereas the no-baseline case showed significant instability and slower improvement.

Average test reward: 158.4 episode length: 158.4

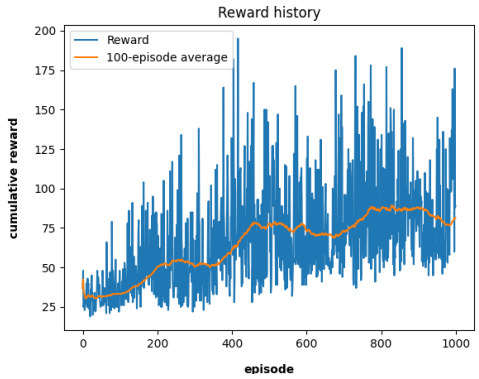

Figure 4: Reward history - Basic REINFORCE with b = 20

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

A normalized baseline falls under the category of state-independent baselines but with an added transformation for stability. It normalizes the rewards (or returns) by centering them around their mean and scaling them by their standard deviation:

$$\text{Normalized Rewards} = \frac{R - \text{mean}(R)}{\text{std}(R) + \epsilon}$$

where ϵ is a small constant to avoid division by zero.

Average test reward: 193.6 episode length: 193.6 Average test reward: 134.6 episode length: 134.6 Average test reward: 186.0 episode length: 186.0

This type of baseline doesn't directly depend on the state, but it helps stabilize training by reducing the range and variance of the rewards, which is especially useful when raw rewards vary significantly.

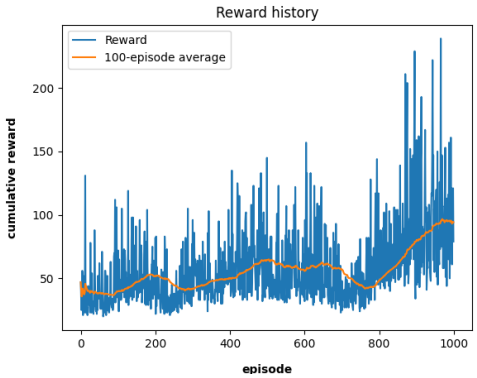

Figure 5: Reward history - Basic REINFORCE with discounted rewards normalized to zero mean and unit variance,

5.4 Actor-Critic Methods

Now it's been tested the two actor critics methods presented before, the PPO (Figure) 6) and SAC (Figure 7).

Comparing them over the same number of epochs Both graphs shows a improvement over the number of epochs bu the final result is not the same. In fact, you see that the poc achieved a lower final reward value and the path is not linear as the sac method, it has its own spikes and decreases. On the other hand the sac after the first 5000 epochs where the rewards has found difficulties to increase it started to increase linearly achiveving a high final reward. These observations found a practical view when testing the model obtained seeing that the second is obviously much better than the second one.

Speaking of time comsumption

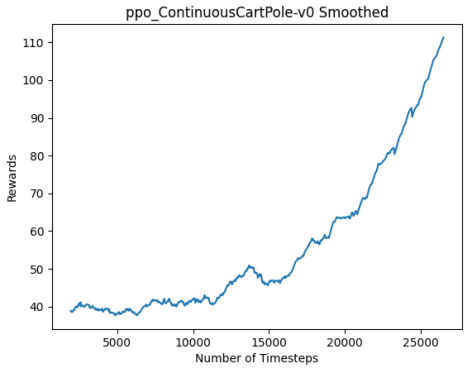

Figure 6: Reward history - PPO

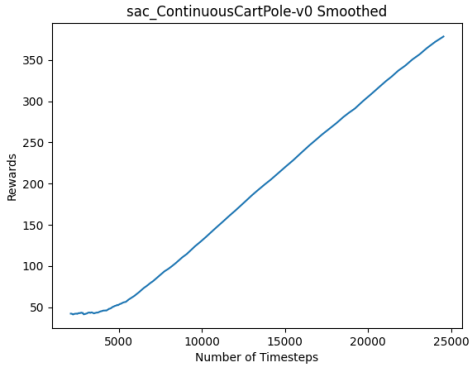

Figure 7: Reward history - SAC

5.5 Comparing all methods

5.6 Results

Compare Terms of reward and time consumption. highlighting any notable differences in terms of learning stability and convergence speed Additionally, compare the results with the REINFORCE

5.7 Hyperparameter modification

adjust any hyperparameter such as learning rate or others as presented in the documentation if needed.

References

[1] URL https://www.gymlibrary.dev/environments/classic_control/cart_pole/.