Reinforcement Learning Assignment 3: Policy-based RL

Georgios Doukeris (s3536149) Evan Meltz (s3817911) Giulia Rivetti (s4026543)

Abstract

In this assignment, we explore policy-based reinforcement learning techniques, which, as opposed to the value-based methods that we have already studied, directly optimize the policy function without explicitly estimating the value function. In particular, we have implemented two policy-based algorithms: REINFORCE and Actor Critic. Then, we have performed hyperparameter tuning, as well as several experiments on those two algorithms, to asses their performance and how they are affected by regularization techniques and strategies that help stabilize the variance.

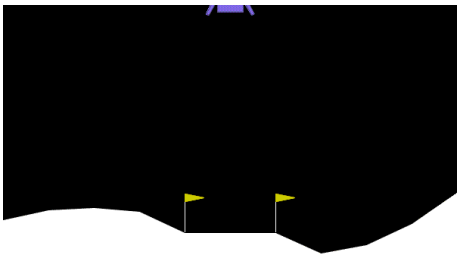

Figure 1. Lunar Lander environment from OpenAI's Gym library. The image depicts a space craft (purple object on top) that is trying to land without crushing on the white surface.

1. Introduction

For the final assignment in this course, contrary to the value-based reinforcement learning done in the previous assignment, we look into the policy-based approach where the policy is optimized directly as oppose to deriving it from the representation of the value function. To do this, we decided to choose the Lunar Lander environment, where the objective is to guide a rocket to land within a landing pad. First we look into the REINFORCE algorithm (Section 4), a policy based algorithm where the agent bases its action in various states on a policy which is derived from a neural network. Then, in section 5, we look into multiple variants of the Actor-Critic algorithm comparing the effects of: bootstrapping; baseline subtraction; and both together on the algorithm. For both algorithms, entropy regularization will be employed. Finally, in section 6 we have run various experiments to compare these algorithms in order to understand their differences, their respective performances and fully investigate these policy-gradient techniques.

2. Environment

As mentioned above, the environment we chose to look at is the Lunar Lander from OpenAI's Gym library. This environment involves a classic rocket trajectory optimization problem where the agent must learn to navigate the rocket to land safely within the designated landing area. There are four actions available for the agent to implement which include: do nothing; fire left orientation engine; fire right orientation engine; or fire main engine. Landing outside of the landing pad is possible and even crashing the rocket. The reward for each step is primarily based on how close to the landing pad the rocket is when landing, with 100 additional points being either added or subtracted depending on whether the rocket is landed safely or crashes (Klimov, 2022).

Environment Description

In the **LunarLander** environment from OpenAI's Gym, the agent controls a two-dimensional spacecraft against a backdrop simulating the moon's surface. The environment incorporates physics dynamics such as gravity, inertia, and collisions. The landscape features a designated flat landing area, highlighted by two flags to aid visual navigation. The spacecraft initiates each episode from a random altitude with varying initial velocities and orientations. The primary objective for the agent is to achieve a safe landing on the specified pad, efficiently managing fuel consumption and avoiding any crash scenarios (Klimov, 2022)

Control Dynamics

The agent has four discrete actions at its disposal:
(a) Do nothing: The spacecraft continues under the influence of its current velocity and gravity.
(b) Fire main engine: This applies an upward thrust, coun-

.

teracting gravity and slowing the descent.
(c) Fire left orientation engine: This generates a counterclockwise rotational force.
(d) Fire right orientation engine: This generates a clockwise rotational force. These controls allow the agent to adjust the spacecraft's position, speed, and orientation effectively, with the environment's physics engine providing realistic reactions to these inputs.

Rewards System

The LunarLander environment employs a strategic scoring system designed to promote efficient and controlled navigation. Landing smoothly on the designated pad grants a significant positive reward. Continuous operation of the main engine incurs a penalty, simulating fuel consumption. Crashing leads to a substantial negative reward. Deviating from the landing pad results in incremental penalties, proportional to the distance from the target.

Learning Challenges

Agents in this environment must balance several critical objectives:
1. Fuel Efficiency: Minimizing fuel use is crucial, as each use of the main engine reduces the score.
2. Stability: Managing the spacecraft's rotation to prevent uncontrolled tumbling.
3. Precision: Accurately navigating to the landing zone despite gravitational influences and initial conditions.
4. Robustness: Adapting to various starting scenarios and disturbances without failing.

Use in Research

The LunarLander environment is extensively utilized in reinforcement learning research as a benchmark to evaluate algorithms capable of managing continuous state and action spaces. It challenges the algorithms to optimize control strategies in scenarios that require handling long-term action consequences and dealing with uncertainty. Such capabilities are imperative for real-world applications like autonomous driving and advanced robotics.

3. Background

Entropy Regularization In deep reinforcement learning, it is important to guarantee exploration, in order to avoid sampling only a fraction of the state space, which can cause the algorithm to get stuck in local optima (Plaat, 2023). For this purpose, we have employed in the REINFORCE algorithm entropy regularization, which indeed provides and incentive to sometimes try an action that seems suboptimal: by incorporating entropy into the loss function, the algorithm encourages policies to exhibit greater exploration and uncertainty, resulting in more versatile actions (Xia, 2024). The strength of entropy regularization can be controlled by the entropy coefficient β. For this assignment, we have used this strategy for both the REINFORCE and the Actor Critic algorithms, showing the effects of its application in Paragraphs 6.2 and 6.3.

4. REINFORCE

REINFORCE is a policy-based algorithm that learns a parametrized policy where an action is selected without consulting a value function; the policy function is represented directly, allowing the selection of continuous actions (Plaat, 2023). In the algorithm, a neural network is employed to present a policy, which guides the actions of the agents in different states. The neural network's parameters are updated based on the obtained rewards, with the goal of favouring actions leading to higher cumulative rewards (GeeksForGeeks, 2024b).

Algorithm The specifics of the REINFORCE algorithm are presented in Algorithm 1. After the initialization of the policy network's parameters, the training loop starts by iterating over a certain number of episodes. Then, for each episode, the environment is reset to get the initial state s_t. During each time-step of an episode, the agent selects an action based on the current state using the policy network $\pi_\theta(a|s)$. After the selected action has been executed, the corresponding state and rewards are observed. Once the episode is terminated, the discounted rewards are computed according to the following formula:

$$R_t \leftarrow \sum_{k=t}^{T-1} \gamma^{k-t} r_k$$

where r_k represents the reward obtained after taking a certain action at a given time-step, and γ is the discount factor, which determines the importance of future rewards, relative to immediate rewards.
After that, the policy loss is computed and then regularized using entropy regularization, in order to encourage exploration, following the formula:

$$\mathcal{L} \leftarrow \mathcal{L}_\theta - \beta \cdot \mathcal{L}_\mathcal{H}$$

Here β is the entropy coefficient, which determines the strenght of entropy regularization; $\mathcal{L}_\theta$ is the policy loss, computed based on the current policy and the discounted rewards:

$$\mathcal{L}_\theta \leftarrow -\sum_{t=0}^{T} log(\pi_\theta(a_t|s_t)) \cdot R_t$$

$\mathcal{L}_\mathcal{H}$ represents the entropy loss, which is computed by taking the mean of the entropy values, which are obtained as

follows:

$$\mathcal{H}(\cdot|s;\theta) = \sum_a \pi_\theta(a|s) log(\pi(a|s;\theta))$$

The gradients of the policy loss are then computed and used to update the parameters of the policy network through backpropagation (gradient ascent):

$$\theta \leftarrow \theta + \alpha\gamma^t R_t \nabla_\theta log(\pi_\theta(a_t|s_t))$$

With this formula, the algorithm is basically pushing the parameters of the policy in the direction of the better action (multiplied proportionally by the size of the estimated action-value) to know which action is best (Plaat, 2023).

Algorithm 1 REINFORCE Algorithm

Inputs:
  Policy $\pi_\theta(a; s)$, learning rate α, entropy coefficient β, discount factor γ, number of episodes M
Initialize:
  Policy parameters θ
for episode = 1, ..., M **do**
  while not done **do**
    Reset environment and get initial state s_t
    Compute the actions probabilities of s_t using $\pi_\theta(a|s)$
    Sample an action a_t based on the action probabilities
    $\mathcal{H}(\cdot|s;\theta) = \sum_a \pi_\theta(a|s) log(\pi(a|s;\theta))$ ▷ Entropy update
  end while
  $R_t \leftarrow \sum_{k=t}^{T-1} \gamma^{k-t} r_k$ ▷ Discounted rewards
  $\mathcal{L}_\theta \leftarrow -\sum_{t=0}^{T} log(\pi_\theta(a_t|s_t)) \cdot R_t$ ▷ Policy loss
  $\mathcal{L}_\mathcal{H} \leftarrow \mathcal{H}.mean()$ ▷ Entropy loss
  $\mathcal{L} \leftarrow \mathcal{L}_\theta - \beta \cdot \mathcal{L}_\mathcal{H}$ ▷ Loss with entropy regularization
  $\theta \leftarrow \theta + \alpha\gamma^t R_t \nabla_\theta log(\pi_\theta(a_t|s_t))$ ▷ Policy network update
end for
Return θ

Architecure As for the architecure of the policy network, it is a simple feed-forward neural network consisting of two fully-connected layers. The hidden layers employ the ReLU activation function, whereas the output layer applies the softmax activation function in order to obtain a probability distribution over actions. The network is characterized by a forward pass, so that the input data is propagated through the layers of the network.

5. Actor Critic

The actor critic algorithm is an enhancement of REINFORCE, introduced in order to solve the high variance problem of that algorithm. Basically, it combines the value-based advantage of low variance through the actor, with the policy-based advantage of low bias via the critic (Plaat, 2023).

In policy based methods, the variance issue is typically a consequence of the high variance of the cumulative reward estimate, and the high variance in the gradient estimate. To address both of this problems, two methodologies were introduced in the actor critic algorithm: *bootstrapping* for better reward estimates, and *baseline subtraction* to lower the variance of gradient estimates (Plaat, 2023).

Bootstrapping In order to address the high variance of cumulative reward estimates, temporal difference bootstrapping can be used, so that the value function is boostrapped step by step. In particular, the value function is used to compute intermediate n-step values per episode, trading-off variance for bias (Plaat, 2023).

Baseline subtraction Another way of decreasing the variance is to subtract a baseline from a set of numbers, a procedure known as baseline subtraction. Since a common choice for the baseline is the value function, this method generally subtracts a value $V(s_t)$ from a state-action estimate $Q(s_t, a_t)$ and in this way it computes the so called *advantage function* $A_t(s_t, a_t)$. This resulting function estimates how much better a particular action is compared to the expectation of a particular state (Plaat, 2023).

Algorithm The actor critic algorithm is illustrated in Algorithm 2. It starts by randomly initializing the policy parameters θ (the actor) and the value function parameters ϕ (the critic). Then, at each time-step of an episode, the agent interacts with the environment by taking an action a_t according to the current policy $\pi_\theta(a|s)$, namely based on the probabilities output by the actor network; in return the agents receives the corresponding observation s_t and reward r_t. Based on the obtained action probabilities, the algorithm also computes the entropy, which will be later used for entropy regularization:

$$\mathcal{H}_\theta(a|s) = \sum_a \pi_\theta(a|s) log(\pi(a|s;\theta))$$

After that, bootstrap is executed by estimating the expected return from the current state using the value function $V_\phi(s_{t+1})$(the critic network) and the obtained reward r_t, thus obtaining the n-step target:

$$\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\phi(s_{t+n})$$

Now, the advantage function or temporal difference error can be computes as:

$$\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$$

After computing the actor and critic losses based on the advantage function, the algorithm updates the actor's parameters θ using the gradient ascent:

$$\theta \leftarrow \theta + \alpha \sum_t \left[\hat{A}_n(s_t, a_t) \cdot \nabla_\theta \, log(\pi_\theta(a_t|s_t)) \right]$$

The policy gradient is derived from the advantage function and guides the actor to increase the probabilities of actions that lead to higher advantages (GeeksForGeeks, 2024a).
The critic's parameters ϕ are updated by computing the gradient of the critic's loss computed before during backpropagation:

$$\phi \leftarrow \phi - \alpha \cdot \nabla_\phi \sum_t \left[\hat{A}_n(s_t, a_t) \right]^2$$

Algorithm 2 Actor critic Algorithm

Inputs:
Policy $\pi_\theta(a|s)$, value function $V_\phi(s)$, estimation depth n, learning rate α, number of episodes M, entropy coefficient β
Initialize:
Randomly initialize θ and ϕ
for episode = 1, ..., M **do**
  Reset the environment and initialize episode reward
  while episode not done **do**
    Select an action a_t according to $\pi_\theta(a|s)$
    $\mathcal{H}_\theta(a|s) = \sum_a \pi_\theta(a|s) log(\pi(a|s;\theta))$ ▷ Compute the entropy
    Take the action a_t and observe state s_t and reward r_t
    $\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\phi(s_{t+n})$ ▷ Bootstrap
    $\hat{A}(s_t, a_t) = \hat{Q}(s_t, a_t) - V_\phi(s_t)$ ▷ Baseline subtraction
  end while
  $\phi \leftarrow \phi - \alpha \cdot \nabla_\phi \sum_t \left[\hat{A}_n(s_t, a_t) \right]^2$ ▷ Descent advantage loss
  $\theta \leftarrow \theta + \alpha \sum_t [A(s_t, a_t) \cdot \nabla_\theta \, log(\pi_\theta(a_t|s_t))]$ ▷ Ascent policy gradient
end for
Return Parameters θ

Architecture The Actor Critic model is characterized by two networks: **Policy network (the actor)**: it consists of a feed-forward neural network with two hidden layers. The first hidden layer is a linear layer with 128 neurons, followed by a dropout layer to prevent overfitting. A Parametric Rectified Linear Unit activation function is then applied to introduce non-linearity.The output layer is another linear layer that produces the output probabilities for each action using a softmax activation function (Uluay, 2023). **Value Network (the critic)**: similarly to the actor network, the critic network also consists of a feed-forward neural network with two hidden layers.The architecture is identical to the Actor network up to the output layer, which consists of a single neuron representing the estimated value of the state (Uluay, 2023). Then, our Actor Critic model combines both architecture into a single model.

6. Experiments

In this section, we present the results of the experiments we have executed in order to asses the performance of our models, as well as understand how they are affected by the usage of methodologies like entropy regularization, n-step bootstrap and baseline subtraction.

6.1. Hyperparameter tuning

The performance of a model can be significantly affected by the choice of hyperparameters, and therefore we have conducted hyperparameter tuning in order to understand which are the best values that we can use to initialize our models. In particular, we have decided to tune the following parameters:
1. Learning rate α: The learning rate controls the magnitude of parameter updates during training and directly impacts the convergence of the neural network. A higher learning rate allows for faster convergence but can cause instability in training. On the other hand, a lower learning rate can lead to more stable learning dynamics but reach convergence more slowly.
2. Discount factor γ: The discount factor γ, influences the importance of future rewards relative to immediate rewards. A higher γ prioritises long-term rewards, encouraging the agent to make decisions that maximise cumulative rewards over time. Instead, a lower discount factor value prioritises more short-term rewards.
3. **Entropy coefficient** β: As we have already explained in paragraph 3, the entropy coefficient β controls the entropy regularization strength and therefore the amount exploration.
We have decided to perform hyperparameter tuning by applying grid search, and we have considered as the best configuration the one yielding to the highest mean reward over 5000 episodes.

Tuning REINFORCE When running hyperparameter tuning for the REINFORCE model, the parameters which performed the best are: learning rate $\alpha = 0.001$, discount factor $\gamma = 0.99$, and entropy coefficient $\beta = 0.01$

Actor critic with bootstrap and baseline subtraction tuning For this model, the best parameters found after hy-

perparameter tuning are: learning rate $\alpha = 0.001$, discount factor $\gamma = 0.99$, and entropy coefficient $\beta = 0.01$

Tuning Actor Critic with Bootstrap For the actor critic model with only bootstrap and without baseline subtraction, the best parameters were found to be: learning rate $\alpha = 0.01$, discount factor $\gamma = 0.99$, and entropy coefficient $\beta = 0.1$

Tuning Actor Critic with Baseline Subtraction In the case of the actor critic model with baseline subtraction and without the bootstrap step, the best parameters we found are: learning rate $\alpha = 0.01$, discount factor $\gamma = 0.99$, and entropy coefficient $\beta = 0.01$

6.2. REINFORCE experiments

In this part of the assignment, we have performed several experiments, in order to assess the performance of the REINFORCE algorithm under some fixed parameters; then, we have also analyzed the effects of entropy regularization on the algorithm, by changing the regularization strength.

REINFORCE performance Figure 2 shows the learning curve of the REINFORCE algorithm obtained after training, with the rewards plotted as a function of the number of episodes. For this experiment, we have considered 5000 episodes and performed 5 independent repetitions to address the stochasticity of the environment.

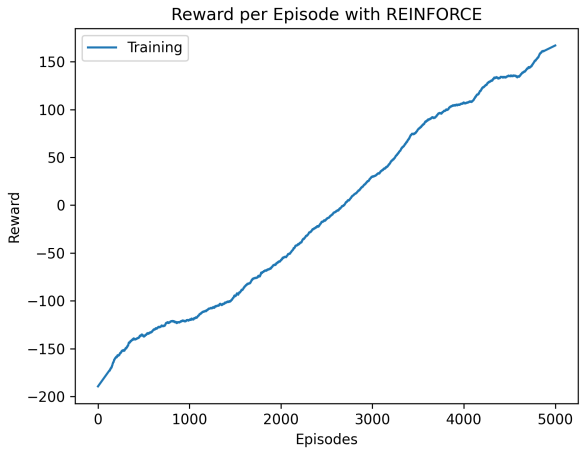

Figure 2. Learning curve of REINFORCE algorithm where the rewards are plotted as a function of the number of episodes. The plot was obtained by executing 5 independent repetitions, each of 5000 episodes.

By looking at the plot we can see that the REINFORCE algorithm overall shows a good performance, since over the episodes, the REINFORCE agent is able to properly learn the Lunar Lander task since increasing values of rewards are obtained episode after episode, achieving rewards up to 150.

We can notice, however, that after 5000 episodes, the algorithm has not yet converged to a stable reward value. Therefore, we have decided to run the algorithms for more repetitions in this case, to understand if the algorithm was actually able to converge. The result is shown in Figure 3, where the rewards are plotted over 11000 repetitions. The

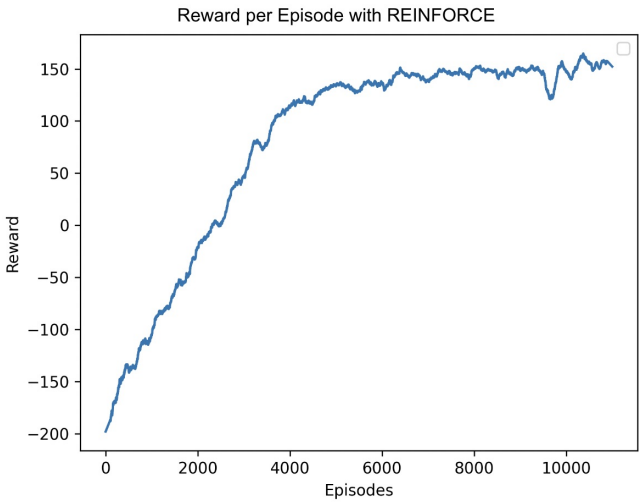

Figure 3. Learning curve of REINFORCE algorithm where the rewards are plotted as a function of the number of episodes. The plot was obtained by executing REINFORCE for 11000 episodes.

learning curve appears to have an oscillating behavior at certain moments, but in the end it is able to slowly converge towards high rewards, indicating that the agent was able to learn a good policy. These two plots clearly show that the learning is slow, and convergence is reached only after many episodes. We attribute this behavior not only to the complexity of the Lunar Lander environment, but also to the high variance issue that characterizes the REINFORCE algorithm. Indeed this method randomly generates a full trajectory, resulting in high variance, and as a consequence policy improvement happens infrequently, thus leading to slow convergence (Plaat, 2023). Therefore, it seems that the policy gradients of the algorithm indeed suffer from high variance. In order to double check this, we have also performed a gradient analysis by displaying the values of the policy gradients to compare them: we have observed that indeed the difference between one gradient value and the other can be quite high sometimes during the algorithm run, confirming the high variance issue that characterizes REINFORCE.

REINFORCE and Entropy Regularization We have conducted an experiment where we compared the learning curves of the REINFORCE algorithm, obtained by employ-

ing different values for the entropy coefficient β, to see its effect on the algorithm's performance.
Figure 4 shows the rewards as a function of the number of episodes obtained with the REINFORCE algorithm by employing four different entropy regularization strengths: 0.001, 0.01, 0.1, and 1.0.

By looking at the plot we can observe that a high value

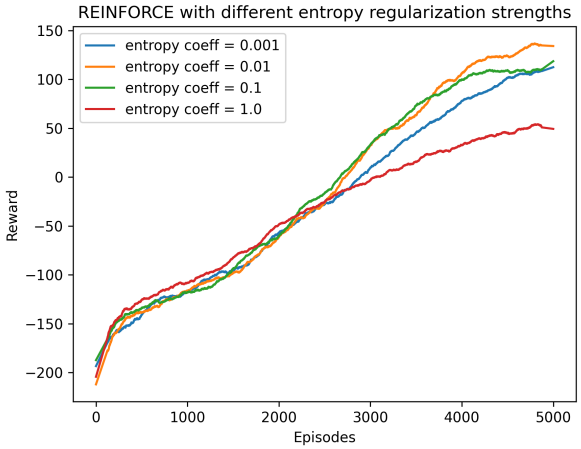

Figure 4. Learning curve of REINFORCE algorithm where the rewards are plotted as a function of the number of episodes. The learning curve shows the behavior for different entropy coefficient values. The plot was obtained by executing 5 independent repetitions, each of 5000 episodes.

for the entropy coefficient, such as 1.0, results in an initial faster learning, but later the learning curve converges to lower values of reward. This is due to the fact that, if the entropy coefficient is too high, the agent favors too much exploration over exploitation, and it is not able to exploit the most rewarding actions effectively. The best results were achieved with a more moderate entropy coefficient: when $\epsilon = 0.01$ there is more balance between exploration and exploitation, but the agent is incentivised to explore more thanks to entropy regularization, allowing the agent to achieve rewards of 150 within 5000 episodes. A value of $\beta = 0.001$, which is a small value resulting in almost no entropy regularization, seems to lead to lower rewards overall, considering 5000 episodes. Therefore, it can be concluded that entropy regularization can be beneficial for the REINFORCE algorithm, promoting different actions and thus preventing it from getting stuck in a suboptimal policy (AI, 2020).

6.3. Actor-Critic experiments

As with REINFORCE in the previous section, here we have performed experiments with the Actor-Critic algorithm using fixed parameters. This analysis focuses on evaluating the impacts of bootstrapping and baseline subtraction individually, as well as their combined effect on the algorithm's performance. Additionally, we explore the consequences of enhancing the algorithm by increasing the number of steps in n-step bootstrapping. Lastly we compare the effects of entropy regularization on the algorithm.

N-Step Bootstrapping The figure illustrates the performance of different N-step methods in terms of reward over a series of episodes. The x-axis represents the number of

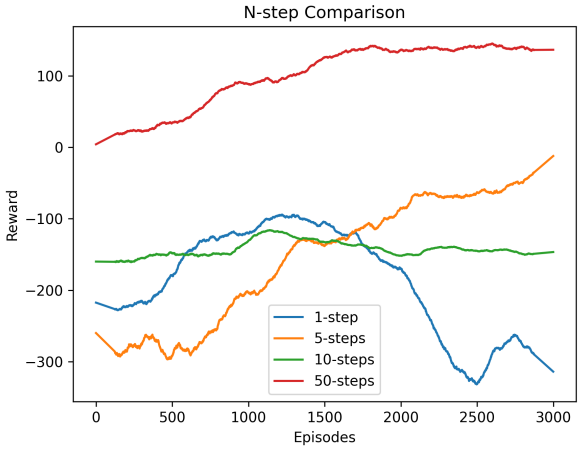

Figure 5. Plot showing the rewards as a function of the number of episodes for the actor critic model with different number of steps n. The plot was obtained by executing 5 distinct repetitions.

episodes, ranging from 0 to 3000, and the y-axis denotes the reward values. The four lines, each corresponding to a different N-step method (1-step, 5-steps, 10-steps, 50-steps), show varying trends in reward accumulation. The red line, representing the 50-step method, consistently achieves higher rewards as episodes increase, indicating a steady and robust performance. In contrast, the 1-step method, shown in blue, experiences significant fluctuations and generally lower rewards, suggesting it may struggle to stabilize. The orange and green lines, for the 5-steps and 10-steps methods respectively, demonstrate intermediate performance with some peaks and troughs but generally trending upwards, implying a balance between stability and responsiveness.In this graph we can clearly see that the algorithms performs best with n-step=50, underlying the importance of the n-step procedure in the actor critic performance.

Actor-Critic and Entropy Regularization As done with REINFORCE, we repeated the experiment of comparing different values of regularization strengths for the entropy coefficient β, to see its impact on the algorithm's performance.

From figure 6, it can be seen without entropy regularization, the policy converges slowly and is more prone to suboptimal

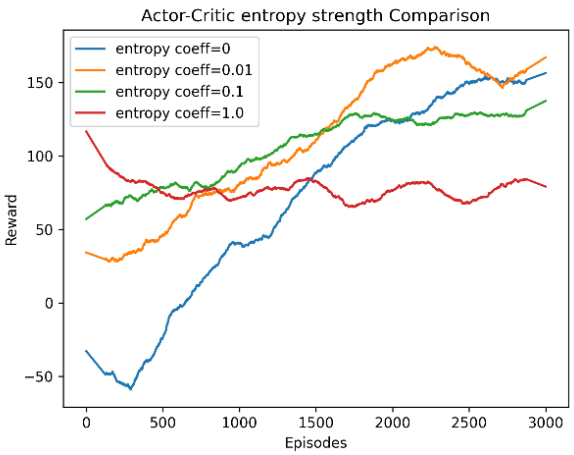

Figure 6. Learning curve of Actor-Critic algorithm where the rewards are plotted as a function of the number of episodes. The learning curve shows the behavior for different entropy coefficient values. The plot was obtained by executing 5 independent repetitions, each of 3000 episodes

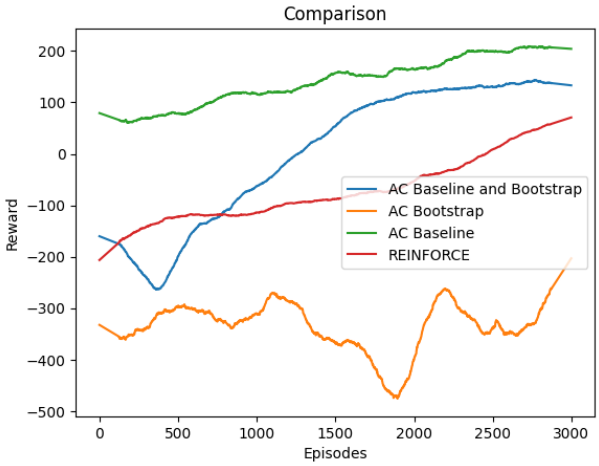

Figure 7. Comparison of algorithms REINFORCE, Actor Critic with Bootstrapping, Actor Critic with Baseline Subtraction and Actor Critic with Bootstrapping and Baseline Subtraction

policies initially and begins with a negative dip. it does demonstrates improvement and converges at a high positive reward, indicating it is still able to effectively learn without entropy regularization.

Extending training to 5000 episodes could provide further insights, as evidenced by the continued upward trend in the curves around the 3000th episode. This suggests that there is still potential for improvement before convergence, potentially leading to higher overall rewards. However due to time and resource constraints this was not possible.

However, from this plot, it appears that the value of 0.01 has the most effective exploration/explotation trade-off and while beginning lower than the higher values, it ultimately reaches a higher reward in the later episodes where it appears the values 0.1 and 1.0 have already converged likely indicating an overemphasis on exploitation and possible entrapment in local optima.

6.4. REINFORCE and Actor Critic Comparison

The graph in Figure 7 illustrates the evolution of rewards collected over 3000 episodes using various reinforcement learning algorithms. On the x-axis, the episodes are enumerated, whereas the y-axis quantifies the accumulated rewards. The graph compares four distinct strategies: ”AC Baseline and Bootstrap” (blue line), ”AC Bootstrap” (orange line), ”AC Baseline” (green line), and ”REINFORCE” (red line). Initially, the actor critic model with both bootstrap and baseline subtraction demonstrates a notable decline in reward, possibly due to exploration or adaptation phase, before it stabilizes and ascends, ultimately achieving the highest rewards among the methods tested. In contrast, the actor critic model employing only bootstrap, after a similar initial dip, follows a steadier upward trajectory but does not reach the heights of the ”AC Baseline and Bootstrap” method, suggesting that the addition of baseline and bootstrap techniques may enhance performance in complex environments. The actor critic model with only baseline subtraction maintains a consistent, moderate gain throughout the episodes, reflecting a potentially less volatile approach but with limited peak reward capability. The ”REINFORCE” method displays considerable variability with a general upward trend, yet it encounters significant fluctuations and a steep drop around the midpoint, indicating a higher sensitivity to episode variance or parameter settings. Each line’s behavior encapsulates the algorithm’s efficiency in learning and adapting over successive episodes, offering insights into the stability and optimization capabilities inherent in each method. In the specific environment, the AC with the baseline significantly outperforms the other three methods.

6.5. Bonus: Proximal Policy Optimization (PPO)

In this part of the assignment, we have decided to experiment with the policy gradient loss PPO (Proximal Policy Optimization). We have applied this algorithm to the actor critic model to make sure that the updates to the actor’s policy were not too drastic; indeed, PPO encourages small adjustments based on what worked before by using a ”trust region” (Uluay, 2023). In particular, PPO uses a clipping function to bound the change in the policy. This function takes the old policy and the new one, and clips the new policy if it is too far from the old policy. This helps to ensure that the new policy is not too different from the old one, which helps to prevent instability (Santhosh, 2023). Figure

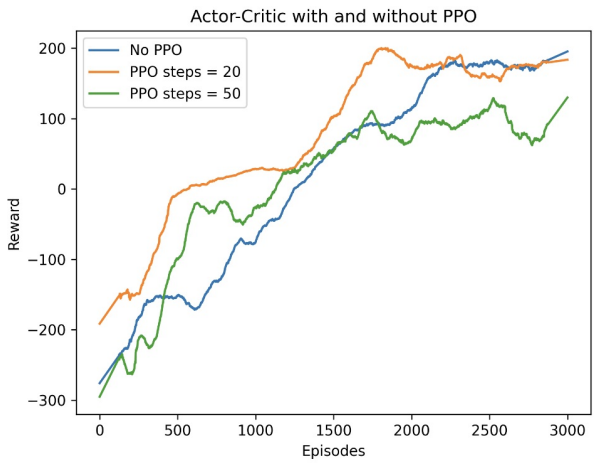

Figure 8. Learning curve of the Actor Critic algorithm where the rewards are plotted as a function of the number of episodes. One learning cuve shows the case where no PPO was applied, whereas the other two were obtaiend by executing 20 and 50 steps in the PPO procedure.

8 shows the comparison between the learning curves of the actor critic model with and without the application of PPO and for two different number of steps in the PPO procedure (20 and 50); as for the other figures, the rewards are plotted as a function of the number of executed. The plot shows that with the application of PPO for 20 steps, the algorithm is able to learn faster and converge earlier to higher rewards, arriving up to 200, meaning that the agent was indeed able to learn the Lunar Lander task. However, by looking at the plot with 50 number of steps in the PPO procedure, we can notice that the learning is more oscillating with respect to the case where no PPO was applied and moreover the final rewards reached are lower. This indicates that the number of steps in the PPO procedure is an important hyperparameter that needs to be tuned properly, otherwise it could lead to lower performance. Therefore, PPO could be beneficial in algorithms like the actor critic, stabilizing and accelerating the learning process, but it introduces an additional hyperparameter that needs to be properly tuned.

7. Conclusions

The primary goal of this project was to implement two policy-based reinforcement learning algorithms within a Gym environment. Through these tasks, we gained a deeper understanding of how the REINFORCE algorithm functions and explored the Actor-Critic method along with three variations: Bootstrapping, Baseline Subtraction, and a Combination of both, applied to the 'Lunar Lander' environment provided by Gym.

Our hands-on experience demonstrated that these techniques could significantly enhance the stability and facilitate convergence to the optimal policy of the algorithms implemented. We observed that Bootstrapping helps lower variance by refining the accuracy of value function estimates, whereas Baseline Subtraction reduces variance by mitigating the impact of a state's absolute value on updates. The integration of both techniques was found to further diminish variance. In our experiments with both REINFORCE and Actor-Critic, we applied hyperparameter optimization to improve performance. Additionally, we encountered the challenge of explosive gradients; to address this, we implemented gradient clipping.

Furthermore, we evaluated the performance of another algorithm, PPO, within the same environment to compare its effectiveness.

The chosen environment, 'Lunar Lander,' was complex and initially challenging to grasp. However, we successfully comprehended its mechanics and adapted the algorithms to function effectively under these specific conditions (Razvan Pascanu, 2013).

References

AI, S. Entropy Regularization, 2020. URL https://serp.ai/entropy-regularization/.

GeeksForGeeks. Actor-Critic Algorithm in Reinforcement Learning, 2024a. URL https://www.geeksforgeeks.org/actor-critic-algorithm-in-reinforcement-learning/.

GeeksForGeeks. REINFORCE Algorithm, 2024b. URL https://www.geeksforgeeks.org/reinforce-algorithm/.

Klimov, O. Lunar Lander - Gym Documentation, 2022. URL https://www.gymlibrary.dev/environments/box2d/lunar_lander/.

Plaat, A. (ed.). *Deep Reinforcement Learning*. Springer Nature, 2023.

Razvan Pascanu, Tomas Mikolov, Y. B. On the difficulty of training recurrent neural networks, 2013. URL https://arxiv.org/pdf/1211.5063.

Santhosh, S. Reinforcement Learning (Part-8): Proximal Policy Optimization(PPO) for trading environment(TensorFlow), 2023. URL https://medium.com/@sthanikamsanthosh1994/reinforcement-learning-part-8-proximal-policy-optimization-ppo-for-trading-9f1c3431f27d.

Uluay, H. H. Exploring Reinforcement Learning: A Hands-on Example of Teaching OpenAI's Lunar Lander to Land Using Actor-Critic Method with Proximal Policy Optimization (PPO) in PyTorch, 2023. URL https://medium.com/@hasanhuseyinuluayy/exploring-reinforcement-learning-a-hands-on-example-of-teaching-openais-lunar-lander-to-lar

Xia, Z. Coding PPO from Scratch with PyTorch (Part 4/4), 2024. URL https://medium.com/@z4xia/coding-ppo-from-scratch-with-pytorch-part-4-4-4e21f4a63e5c.