Reinforcement Learning 2024
Assignment 3: Policy-based RL

Ritesh Sharma (4158059)

Abstract

In this paper, we implement and analyze Policy-based algorithms on OpenAI Gym's LunarLander-v2 environment. This implementation included the REINFORCE algorithm as well as variants of the actor-critic algorithm. We have experimented with different values for multiple parameters and compare how the algorithms perform.

1. Introduction

The purpose of this study is to provide a comprehensive review of the Lunar Lander environment. The Lunar Lander environment presents a physics-based simulation to optimize rocket landings using principles derived from Pontryagin's maximum principle. This paper includes an in-depth analysis of the simulation's action and observation spaces as well as the reward mechanisms that guide the learning algorithms. We will also experiment with regards to all the hyperparameters and see which ones will work better with regards to this problem.

2. Environment

The Lunar Lander environment utilizes PyGame for rendering which provides a visual representation of the simulation. These technologies create a realistic and dynamic simulation space where different landing scenarios can are tested and analyzed.

2.1. Lunar Lander

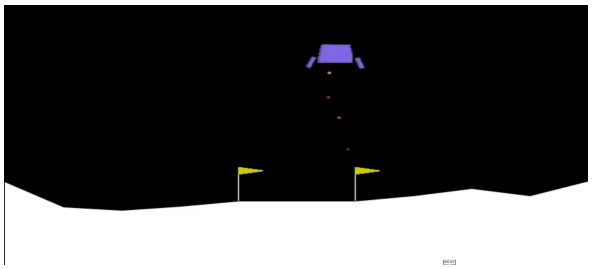

Figure 1. The Lunar Lander environment (OpenAI, 2016–)

2.2. Action Space

The action space in the Lunar Lander environment consists of four discrete actions: do nothing, fire the left orientation engine, fire the main engine, and fire the right orientation engine.

2.3. Observation Space

The observation space in the Lunar Lander environment is an 8-dimensional vector that includes the lander's x and y coordinates relative to the landing pad, x and y velocities, angle and angular velocity, and two booleans indicating whether the left and right legs are in contact with the ground, respectively.

2.4. Rewards

In the Lunar Lander environment, the reward system is structured to promote precise navigation and safe landing. Reward for moving from the top of the screen to the landing pad and coming to rest is about 100-140 points. If the lander moves away from the landing pad, it loses reward. If the lander crashes, it receives an additional -100 points. If it comes to rest, it receives an additional +100 points. Each leg with ground contact is +10 points. Firing the main engine is -0.3 points each frame. Firing the side engine is -0.03 points each frame. Successfully completing the mission, awards the agent +200 points.

2.5. Episode Termination

An episode in the Lunar Lander simulation terminates if the lander crashes by contacting the moon, exits the viewport with an x-coordinate greater than 1, or enters a non-awake state where it neither moves nor collides with other bodies, as defined in the Box2D documentation.

3. Policy-Based Reinforcement Learning

Policy-based reinforcement learning focus directly on modeling and optimizing the policy function, rather than deriving it indirectly through a value function. In policy-based methods, the policy, which specifies the probability distribution of selecting each action given a state, is parameterized

and optimized to maximize the long-term rewards from the environment. This approach is particularly effective for problems with high-dimensional or continuous action spaces. We will be using various policy-based reinforcement learning techniques, including REINFORCE and different actor-critic methods to solve the Lunar Lander problem.

4. REINFORCE

REINFORCE is a Monte Carlo variant of policy gradient methods. This method focus directly on optimizing the policy that the agent uses to decide its actions rather than estimating the value of actions or states as in value-based methods.

The key part of the REINFORCE algorithm is the gradient ascent update step for the policy parameters θ as show in the Equation(1):

$$\theta \leftarrow \theta + \alpha\gamma^t R \nabla_\theta \log \pi_\theta(a_t \mid s_t) \tag{1}$$

This formula updates the policy parameters θ to optimize the policy π_θ by increasing the probability of actions that result in higher returns.

Algorithm 1 Monte Carlo policy gradient (REINFORCE)

Input: A differentiable policy $\pi_\theta(a \mid s)$, learning rate $\alpha \in \mathbb{R}^+$, threshold $\epsilon \in \mathbb{R}^+$
Initialization: Initialize parameters $\theta \in \mathbb{R}^d$
repeat
  Generate full trace $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a \mid s)$
  for $t \in 0, \ldots, T-1$ **do**
    $R \leftarrow \sum_{k=t}^{T-1} \gamma^{k-t} \cdot r_k$
    $\theta \leftarrow \theta + \alpha\gamma^t R \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
  end for
until $\nabla_\theta J(\theta)$ converges below ϵ
return Parameters θ

4.1. Methodology

1. **Inputs and Initialization:**
   - The algorithm starts by initializing the parameters of a policy function, $\pi_\theta(a \mid s)$. The parameters, θ, represent the weights of a neural network.
   - The policy determines the probability of taking action a given the state s.
2. **Generating Episodes:**
   - The agent generates a full trace of states, actions and rewards by interacting with the environment according to the current policy.
3. **Calculating Returns and Updating Parameters:**
   - For each timestep in the episode, the algorithm calculates the total discounted return R. It then performs a gradient ascent update on the policy parameters θ using the gradient of the log probability of the action taken, scaled by the calculated return and the learning rate α.
4. **Convergence Check:**
   - The algorithm repeats the process of generating episodes and updating parameters until the gradient of the performance function $J(\theta)$ with respect to the parameters converges below a set threshold ϵ.

This process iteratively adjusts the policy towards higher expected returns, improving the agent's performance in the environment over time.

5. Actor-Critic

The Actor-Critic method in reinforcement learning combines elements from both policy-based and value-based approaches which improves learning efficiency and stability. This method involves two primary components: the actor and the critic.

1. **Actor:** This component is responsible for selecting actions based on a policy that is parameterized by weights.
2. **Critic:** This component evaluates the action taken by the actor by computing the value function.

5.1. Actor-Critic (Bootstrapping)

In Actor-Critic with Bootstrapping, the actor updates its policy parameters θ in the direction suggested by the critic's TD error. This is done using a policy gradient method where the update is proportional to the gradient of the policy's log-probability weighted by the TD error:

$$\theta \leftarrow \theta + \alpha \cdot \sum_t [\hat{Q}_n(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t)] \tag{2}$$

The critic updates its value function estimates using the temporal difference (TD) error, a form of bootstrapping. The TD error is calculated as:

$$\phi \leftarrow \phi - \alpha \cdot \nabla_\phi \sum_t (\hat{Q}_n(s_t, a_t) - V_\phi(s_t))^2 \tag{3}$$

REINFORCE updates its policy based on the total return of complete episodes, which can introduce high variance, whereas Actor-Critic updates its policy more frequently based on the critic's value estimates, reducing variance and allowing more stable learning.

Algorithm 2 Actor-Critic with Bootstrapping

Input: A policy $\pi_\theta(a \mid s)$, a value function $V_\phi(s)$
An Estimation depth n, learning rate α, number of episodes M
Initialization: Randomly initialize θ and ϕ
repeat
  for $i \in 1, \ldots, M$ **do**
    Sample trace $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a \mid s)$
    for $t \in 0, \ldots, T-1$ **do**
      $\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\phi(s_{t+n})$
    end for
  end for
  $\phi \leftarrow \phi - \alpha \cdot \nabla_\phi \sum_t (\hat{Q}_n(s_t, a_t) - V_\phi(s_t))^2$
  $\theta \leftarrow \theta + \alpha \cdot \sum_t [\hat{Q}_n(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t)]$
until $\nabla_\theta J(\theta)$ converges below ϵ
return Parameters θ

5.2. Actor-Critic (Baseline subtraction)

This method combines policy gradient approaches with value function estimation. The baseline subtraction component helps to reduce variance in the policy gradient estimates which leads to more stable and reliable learning.The baseline is used to compute the advantage function as shown in the Equation(4).

$$A(s, a) = Q(s, a) - V(s) \tag{4}$$

where $Q(s, a)$ is the action-value function, and $V(s)$ is the value function that estimates the expected return from state s. The advantage function estimates how much better a particular action is compared to the expectation of a particular state.

Algorithm 3 Actor-Critic with Baseline Subtraction

Input: A policy $\pi_\theta(a \mid s)$, a value function $V_\phi(s)$, learning rate α, number of episodes M
Initialization: Randomly initialize θ and ϕ
while not converged **do**
  for $i = 1$ to M **do**
    Sample trace $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a \mid s)$
    for $t = 0$ to $T-1$ **do**
      $Q(s_t, a_t) = r_t + \gamma V(s_{t+1})$
      $\hat{A}(s_t, a_t) = Q(s_t, a_t) - V_\phi(s_t)$
    end for
    $\phi \leftarrow \phi - \alpha \nabla_\phi \sum_t (\hat{A}(s_t, a_t))^2$
    $\theta \leftarrow \theta + \alpha \sum_t [\hat{A}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t)]$
  end for
end while
return Optimized parameters θ

5.3. Actor-Critic (Bootstrapping and baseline subtraction)

This method combines the techniques of Bootstrapping and Baseline Subtraction. It enhances learning by updating policies and value functions based on incomplete trajectories (from Bootstrapping) and stabilizes these updates by using a baseline to reduce variance in policy updates (from Baseline Subtraction).

Algorithm 4 Actor Critic with Bootstrapping and Baseline Subtraction

Input: A policy $\pi_\theta(a \mid s)$, a value function $V_\phi(s)$
An Estimation depth n, learning rate α, number of episodes M
Initialization: Randomly initialize θ and ϕ
while not converged **do**
  for $i = 1$ to M **do**
    Sample trace $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a \mid s)$
    for $t = 0$ to $T-1$ **do**
      $\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V_\phi(s_{t+n})$
      $\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\phi(s_t)$
    end for
  end for
  $\phi \leftarrow \phi - \alpha \cdot \nabla_\phi \sum_t (\hat{A}_n(s_t, a_t))^2$
  $\theta \leftarrow \theta + \alpha \cdot \sum_t [\hat{A}_n(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t)]$
end while
return Parameters θ

5.4. Methodology

The methodology describes similar and some different steps of different Actor-Critic algorithms.

1. **Inputs and Initialization :** The algorithm defines components: a policy $\pi_\theta(a \mid s)$ that defines the probability of taking action a in state s and a value function $V_\phi(s)$ that estimates the expected return from state s. The learning rate α and the number of episodes M. Parameters θ and ϕ are randomly initialized which parameterize the policy and the value function respectively.

2. **Episode Sampling and Processing :** For each episode, up to M, a complete trace is generated by following the policy $\pi_\theta(a \mid s)$ from the initial state to termination. This trace captures the agent's experience throughout the episode.

3. **Update Calculations :**

   Actor-Critic with Bootstrapping : For each timestep, the bootstrapped estimate $\hat{Q}_n(s_t, a_t)$ is calculated by summing the expected rewards for the next n steps, each discounted based on its temporal distance from

the current state, and then adding the discounted value of the future state n steps later.

Actor-Critic with Baseline Subtraction : For each timestep t within the episode, calculate the action-value $Q(s_t, a_t)$ by adding the immediate reward receivedr_t, to the discounted value of the subsequent state $\gamma V(s_{t+1})$. Compute the advantage $\hat{A}(s_t, a_t)$ by subtracting the value function $V_\phi(s_t)$, which estimates the expected return from state s_t, from the action-value function $Q(s_t, a_t)$, which represents the expected return for taking action a_t in state s_t.

Actor-Critic method with Bootstrapping and Baseline Subtraction : For each timestep, the bootstrapped estimate $\hat{Q}_n(s_t, a_t)$ is calculated by summing the expected rewards for the next n steps, each discounted based on its temporal distance from the current state, and then adding the discounted value of the future state n steps later. For each timestep, the advantage $\hat{A}_n(s_t, a_t)$ is calculate by subtracting the value function's current estimate of the state $V_\phi(s_t)$ from the bootstrapped estimate $\hat{Q}_n(s_t, a_t)$.

4. **Parameter Updates :**

   Actor-Critic with Bootstrapping : The equation (2) shows how the policy parameters θ is updated to increase the probability of actions that leads to higher returns, using the policy gradient weighted by the bootstrapped estimates. The equation (3) shows how the the parameters ϕ of the value function is updated to reduce the mean squared error between the bootstrapped estimates and current estimates

   Actor-Critic with Baseline Subtraction : Gradient descent is performed to minimize the squared advantage as shown in the Equation(5) below :

$$\sum_t \left(\hat{A}(s_t, a_t)\right)^2 \tag{5}$$

   After that gradient ascent is performed to update the policy parameters θ by optimizing the log-probability of the actions taken, weighted by the calculated advantages as shown in the Equation(6) below :

$$\sum_t \left[\hat{A}(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right] \tag{6}$$

   Actor-Critic method with Bootstrapping and Baseline Subtraction : The parameters are tuned to minimize the discrepancy between the bootstrapped values and the current predictions by modifying ϕ. This is achieved by reducing the squared discrepancy of the advantage estimates $\hat{A}_n(s_t, a_t)$ across all timesteps. The parameters θ are tuned to optimize the policy's performance, using the advantage estimates $\hat{A}_n(s_t, a_t)$. The updates are made by applying a gradient ascent method to the policy parameters.

5. **Convergence Check :** The process is repeated until the changes in policy parameters θ converge below a pre-defined threshold ϵ, indicating that the learning has stabilized and the policy has been optimized.

6. **Return Optimized Parameters :** Upon reaching convergence, the optimized policy parameters θ are returned, which are tuned to make effective decisions across different states within the environment.

6. Exploration Strategies

To balance exploration and exploitation we have used entropy regularization. Entropy regularization is an exploration technique that integrates entropy into the policy optimization process to promote exploration by the agent.

Entropy measures the randomness in the probability distribution over the actions chosen by a policy. The entropy can be defined as:

$$H(\pi(\cdot|s)) = -\sum_a \pi(a|s)\log(\pi(a|s)) \tag{7}$$

Entropy is calculated by summing all possible actions a, where $\pi(a|s)$ is the probability of taking action a given state s and $\log(\pi(a|s))$ is the natural logarithm of this probability. The sum is then multiplied by a negative sign ensuring that entropy is a non-negative quantity. High Entropy indicates that the probability distribution over actions is more uniform. Low Entropy indicates that the probability distribution is skewed towards particular actions.

In policy gradient methods, the objective is to maximize the expected return from a policy. Entropy regularization modifies this objective to maintain higher entropy. The modified objective function can be expressed as:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] + \beta H(\pi) \tag{8}$$

Here, $\mathbb{E}_{\tau\sim\pi}[R(\tau)]$ represents the expected return under the policy π, $H(\pi)$ denotes the entropy of the policy, and β is a coefficient that controls the trade-off between maximizing returns and maintaining entropy. This encourages the policy to explore more actions, thus enhancing exploration and preventing premature convergence on suboptimal policies.

7. Results

7.1. REINFORCE: Hyper-parameter Tuning

7.1.1. Learning Rate (α)

In this section, we aim to find the best value of learning rate possible for REINFORCE. We tried three different

Learning Rates (α) [0.001, 0.0005, 0.0001] to see how well it performed.

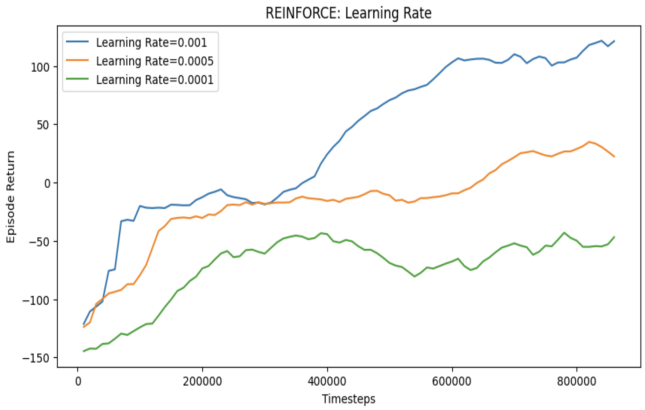

Figure 2. REINFORCE algorithm performance at different values of learning rate. The total number of timesteps are 1 million where the policy used is entropy regularization. Smoothing window = 11.

Based on Figure 2, we can see that REINFORCE algorithm on lunar lander works best when the learning rate = 0.001. The learning seems to be stable and quick. The episode returns are the highest when compared to the other values of learning rate.

7.1.2. Entropy Regularization Strength

In this section of REINFORCE, we aim to find the best value of Entropy coefficient. We tried REINFORCE algorithm with three different Entropy coefficient values [0.1, 0.01, 0.001].

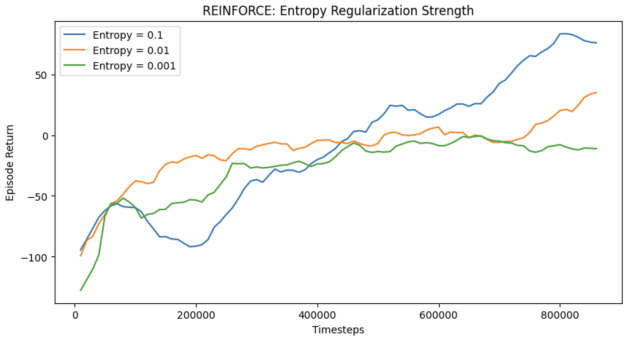

Figure 3. REINFORCE algorithm performance with different values of Entropy coefficient. The total number of timesteps are 1 million where the policy used is entropy regularization. Learning rate at 0.001. Smoothing window is at 11.

From Figure 3, we can observe that the Entropy coefficient at 0.1 performs the best. REINFORCE algorithm can learn quickly with good episode returns. Performance of reinforce on lunar lander is similar for other two values, but entropy coefficient = 0.1 provides higher episode returns.

7.1.3. Discount Factor (γ)

Our goal in this part is to find the best value for the discount factor (γ). We tested how well the REINFORCE algorithm worked with three different values of Gamma (γ): [0.99, 0.95, 0.9]

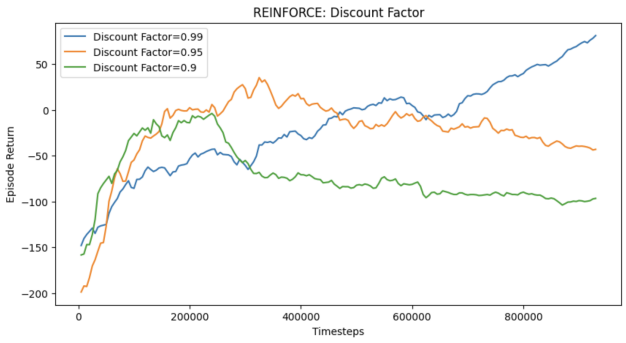

Figure 4. REINFORCE algorithm performance at different discount factor values. The total number of timesteps are 1 million where the policy used is entropy regularization. Learning rate at 0.001. Entropy coefficient is 0.1. Smoothing window is at 11.

From Figure 4, we can observe that the (γ) value at 0.99 performs best and provides better episodic returns for REINFORCE algorithm implementation. REINFORCE at (γ) = 0.99 takes time to learn but its performance is more stable. Therefore, the discount factor (γ) selected is 0.99.

7.2. REINFORCE

In this section we ran the REINFORCE algorithm on lunar lander for 3 million timesteps with tuned hyperparameters. This is to observe the episodic returns for longer runs.

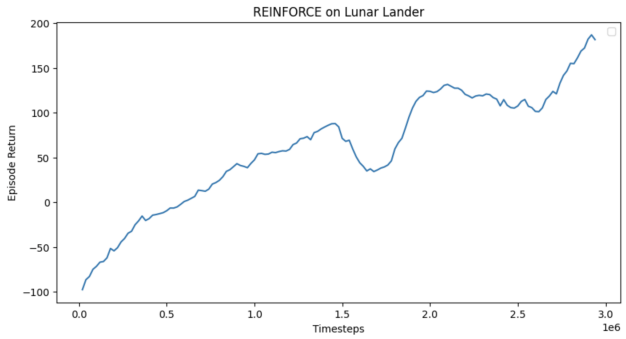

Figure 5. REINFORCE performance for 3 million timesteps where learning rate = 0.001, Entropy coefficient = 0.1 and discount factor = 0.99. Smoothing window is at 13.

We can see from Figure 5 that the learning achieved has high variance but still episode returns are able to go from -100 to almost +200. There is a drop at 1.5 million timestep but the algorithm is able recover well. With learning rate, entropy

coefficient and discount factor tuned this is the highest result we can get for the implementation.

7.3. Actor-Critic: Hyper-parameter Tuning

7.3.1. Learning Rate (α)

In this section of Hyper-parameter Tuning, we aim to find the best value of learning rate possible for Actor-Critic algorithm. We tried three different Learning Rates (α) [0.001, 0.0005, 0.0001] to see how well it performed.

Figure 6. Actor-critic algorithm performance at different values of learning rate. The total number of timesteps are 1 million where policy is entropy regularization. Smoothing window is 11.

Based on Figure 6, we can see that Actor-critic algorithm on lunar lander works best when the learning rate = 0.001. The learning seems to be quick but has variance for all three values. The episode returns are the highest when compared to the other values of learning rate.

7.3.2. Entropy Regularization Strength

In this section, we aim to find the best value of Entropy coefficient. We tried Actor-critic algorithm with three different Entropy coefficient values [0.1, 0.01, 0.001].

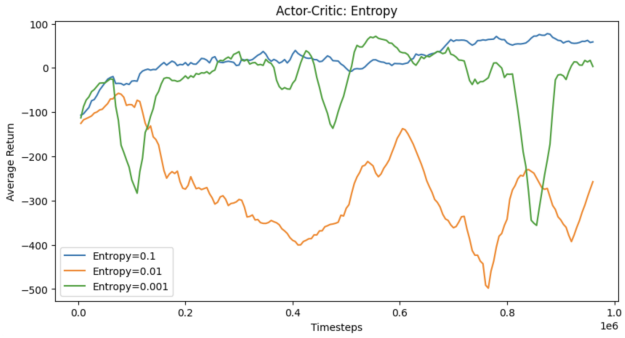

Figure 7. Actor-critic algorithm performance with different values of Entropy coefficient. The total number of timesteps are 1 million where the policy used is entropy regularization. Learning rate at 0.001. Smoothing window is at 11.

From Figure 7, we can observe that the Entropy coefficient at 0.1 performs the best. Actor-critic algorithm can learn quickly with good episode returns but still with very high variance. Performance of Actor-critic on lunar lander is low for 0.0001 and 0.001, but entropy coefficient = 0.1 provides better and positive episode returns.

7.3.3. Discount Factor (γ)

Our goal in this part is to find the best value for the discount factor (γ). We tested how well the Actor-critic algorithm worked with three different values of Gamma (γ): [0.99, 0.95, 0.9]

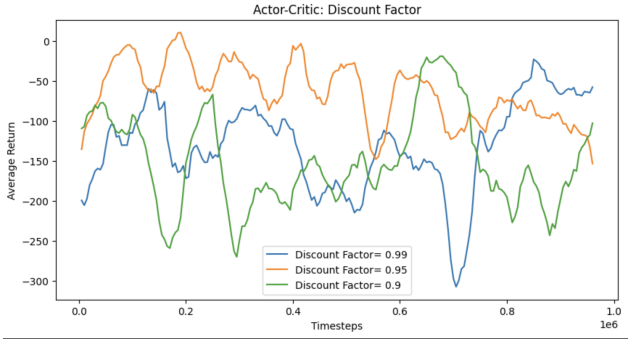

Figure 8. Actor-critic algorithm performance at different discount factor values. The total number of timesteps are 1 million where the policy used is entropy regularization. Learning rate at 0.001. Entropy coefficient is 0.1. Smoothing window is at 11.

From Figure 8, we can observe that the (γ) value at 0.95 performs best and provides better episodic returns for Actor-critic algorithm implementation. Actor-critic at (γ) = 0.95 like all values has very high variance and takes time to learn but still performs better than other two. Therefore, the discount factor (γ) selected is 0.95.

7.3.4. N-Step

Our goal in this part is to find the best value for the N-step. We tested how well the Actor-critic algorithm worked with three different values of N-step: [1, 5, 10]

From Figure 9, we can observe that the Actor-critic performance is very similar for 1-step and 5-step. The performance with 5-step yields slightly better episodic returns. 10-step seems to give very low performance. Therefore, the 5-step selected.

7.4. Actor-Critic Variants

In this section, we are going to run Actor-Critic on its three variants [Bootstrapping, Baseline subtraction, Bootstrapping + baseline subtraction] for 1.5 million timesteps to see which variant performs better and provides stable learning.

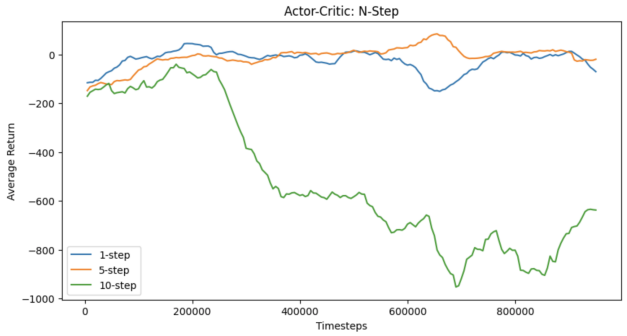

Figure 9. Actor-critic algorithm performance at different N-step values. The total timesteps are 1 million where the policy is entropy regularization. Smoothing window = 11.

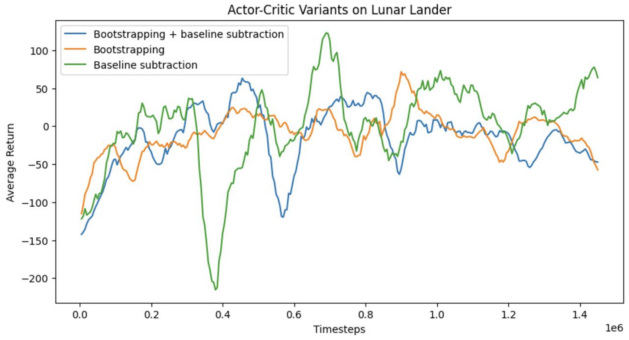

Figure 10. Actor-critic algorithm variants performance. The total timesteps are 1.5 million where the policy is entropy regularization. Smoothing window is at 11.

In Figure 10, all three variants exhibit high variance. Baseline subtraction shows the highest variance but also the highest episodic average returns. Bootstrapping is more stable with the lowest variance, though its returns are lower. Combining bootstrapping with baseline subtraction balances lower variance with high, positive returns.

7.5. Experiment

We compare the performance of REINFORCE and Actor-critic agents. The total timesteps taken will be 1 million with evaluation done at every 5000 timestep. Table 1 provides list of tuned hyper-parameters for REINFORCE and Table 2 does the same for Actor critic Algorithm. The variant used for Actor-critic is Bootstrapping + baseline subtraction.

Parameter	Value
Learning Rate	0.001
Discount Factor	0.99
Entropy Coefficient	0.1
Evaluation	At every 5000 timestep

Table 1. Optimal hyper-parameters for REINFORCE algorithm.

Parameter	Value
Learning Rate	0.001
Discount Factor	0.95
N-step	5
Entropy Coefficient	0.1
Evaluation	At every 5000 timestep

Table 2. Optimal hyper-parameters for Actor-Critic algorithm.

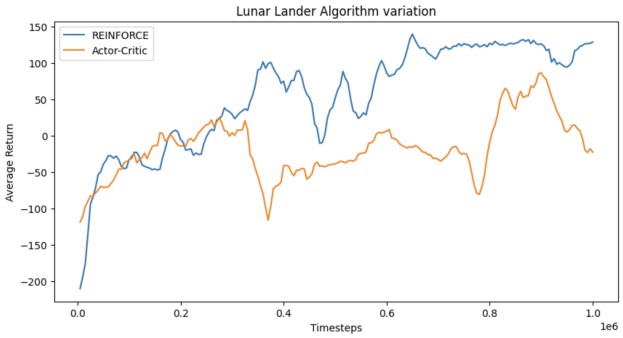

Figure 11. Lunar Lander on both algorithms. The total timesteps are 1 million where the policy is entropy regularization. Smoothing window is at 11.

In Figure 11, we can observe that performance of REINFORCE is better for the given timesteps compared to Actor-critic. Learning for both the algorithm have alot of variance but still both the algorithms reach positive reward. REINFORCE is also quick to learn and achieve results where as Actor-critic takes it time to achieve similar results.

8. Observation

8.1. REINFORCE

Yes, the policy gradients used by the REINFORCE algorithm do indeed suffer from high variance. This is primarily because REINFORCE updates the policy solely based on the returns from complete episodes. It calculates the gradient of the policy's performance by sampling entire episodes and using these samples to estimate the expected return. Each update is weighted by the total return from start to finish of an episode. Since the returns from entire episodes can vary greatly as some episodes may end well with high returns while others could be very bad with low returns due to the actions taken, initial states and the stochastic nature of the environment. This high variability in returns between episodes leads to significant fluctuations in the gradient estimates and as a result, the REINFORCE algorithm suffers from high variance (Sutton & Barto, 2018).

8.2. Actor-Critic effects on policy gradients Variance

- **Bootstrapping:** Bootstrapping reduce the variance of policy updates because it reduces the dependency on the full trajectory outcomes which can be highly variable. By relying on partial trajectories, the updates are less inclined to the fluctuations that might occur in longer sequences and helps to reduce variance.
- **Baseline Subtraction:** Baseline Subtraction use a baseline value which helps to reduce the variance of the policy gradient estimates without biasing the gradients because the baseline normalizes the rewards and effectively centers the gradient updates around zero.
- **Bootstrapping and Baseline Subtraction:** Both bootstrapping and baseline subtraction techniques reduce variance in different way respectively. Bootstrapping limits the scope of the reward sequence considered in each update, reducing the variance associated with longer and more variable trajectories whereas Baseline subtraction normalizes these updates against a baseline, further reducing the impact of outlier rewards or unusual sequences. Thus, both techniques help in the significant reduction of variance.

All three techniques in the Actor-Critic framework reduce the variance of policy gradients differently (Wu & Rajeswaran, 2018).

8.3. Performance

Theoretically, the combination of bootstrapping and baseline subtraction in an Actor-Critic framework should offer a balance between stability and performance by reducing variance and normalizing updates for consistent learning. However, in the Lunar Lander environment, the empirical results present a different narrative.

In comparing REINFORCE with the Actor-Critic variants, specifically the Actor-Critic with Bootstrapping + Baseline Subtraction, it was observed in the analysis that REINFORCE performed better in terms of learning speed and overall returns. While the combination of Bootstrapping and Baseline Subtraction theoretically should perform well by optimizing learning stability and return, in this environment, REINFORCE outperformed the Actor-Critic variants in achieving faster and more effective results.

8.4. Entropy Regularization effects on performance

- Entropy regularization encourage a policy to explore diverse actions, reducing the chances of premature convergence to suboptimal deterministic policies. This leads to the discovery of more effective strategies and improves the robustness of the learned policy. Thus ensuring better performance in different environments.
- Entropy regularization helps prevent the policy from getting stuck in local minima by encouraging exploration of a wider range of state-action pairs. Additionally, it stabilizes training by smoothing learning updates, which are less prone to fluctuations.

9. Discussion

Future research could include an agent with a higher dimensional action space in order to mimic real world events. Also, staying in the realm of policy gradient methods, we believe different algorithms like A2C and A3C could be implemented for this environment. The latter was implemented in this paper (Mnih et al., 2016) yielding positive results, hence this could also be beneficial in our environment. Furthermore, our policy network has one hidden layer of size 256. Future research could include different sizes and more layers for the network.

10. Conclusion

To conclude, we have obtained many results with some more positive than others. We have shown that the gradients suffer from high variance. A surprising find was that for all three variants of the actor critic model showed a lot of variance, and that none of them clearly outperformed the other. We also expected the actor critic model to perform better than the REINFORCE model, however it turned out that was not the case. We believe this case is specific for the lunar lander environment. We achieved positive evaluation rewards for both algorithm proving that the agents are in fact learning.

References

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning, 2016.

OpenAI. OpenAI Gym. https://www.gymlibrary.dev/environments/box2d/lunar_lander/, 2016–.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT press, 2018.

Wu, C. and Rajeswaran, D. Variance reduction for policy gradient with action-dependent factorized baselines. In *Proceedings of the International Conference on Learning Representations*, 2018.