COMPSCI 687 Final Project - Fall 2024

Pranav Jain (33982032) **prajain@umass.edu**
Gargee Shah (34028128) **gmshah@umass.edu**

In this project, we implemented the REINFORCE with Baseline and One-Step Actor-Critic algorithms and conducted experiments on three distinct Markov Decision Processes (MDPs): Cartpole, Acrobot, and Lunar Lander. The performance of each algorithm was analyzed across these environments to assess their effectiveness and convergence.

1 Markov Decision Process (MDP)

1.1 CartPole Markov Decision Process (MDP)

The CartPole problem is a classic reinforcement learning environment, often used to demonstrate fundamental concepts in decision-making under uncertainty. The objective is to balance a pole on a cart by applying forces to the cart. Below is a detailed description of the CartPole as a Markov Decision Process (MDP):

An MDP is defined by a tuple (S, A, P, R, γ), where:

1.1.1 State Space (S):

The state of the CartPole system is represented as a continuous vector with four dimensions. Specifically, the state of the CartPole environment is a 4-dimensional vector, representing the following parameters:

- **Cart Position:** Horizontal position of the cart on a one-dimensional track.
- **Velocity of the Cart:** Rate of change of the cart's position.
- **Pole Angle:** Angle of the pole relative to the vertical.
- **Angular Velocity of the Pole:** Rate of change of the pole's angle.

While the state space is continuous, the environment is often bounded:

$$x \in [-x_{\max}, x_{\max}], \quad \dot{x} \in [-\dot{x}_{\max}, \dot{x}_{\max}]$$

If these bounds are violated, the episode terminates, as the cart is considered to have failed to balance the pole.

1.1.2 Action Space (A):

The action space in the CartPole environment usually consists of two discrete actions:

- $a = 0$: Apply a force on the cart to the left.
- $a = 1$: Apply a force on the cart to the right.

The magnitude of the force is typically constant (e.g., $F = \pm 10$ N).

1.1.3 Transition Dynamics ($P(s'|s, a)$):

The environment is governed by the laws of physics. The next state (s') is computed based on the current state (s) and action (a) using equations derived from Newtonian mechanics:

- The force applied to the cart affects the cart's acceleration and velocity.
- The torque on the pole affects its angular velocity and position.

These dynamics are modeled as deterministic but can be made stochastic by introducing noise.

1.1.4 Reward Function ($R(s, a)$):

The agent receives a reward of:

- +1 for every time step it successfully balances the pole.
- 0 if the episode ends (due to exceeding position or angle limits).

The goal is to maximize the total cumulative reward over the course of an episode.

1.1.5 Discount Factor (γ):

The discount factor determines the agent's focus on future rewards. It is typically set close to 1 (e.g., $\gamma = 0.99$) to encourage long-term balancing.

1.1.6 Initial State:

Each observation is initialized with values uniformly random in the range $[-0.05, 0.05]$, represented as:

$$S_0 = (x, \dot{x}, \theta, \dot{\theta})$$

1.1.7 Terminal State:

An episode terminates when one of the following conditions is met:

- The pole angle exceeds a predefined threshold: $[-12°, 12°]$.
- The cart position moves outside a predefined range: $[-2.4, 2.4]$.

1.1.8 Objective

The agent's goal is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative reward:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

This translates to balancing the pole for as long as possible within the constraints of the environment.

1.1.9 Simulation Environment

For the project, the CartPole is implemented using the Gym framework, specifically the CartPole-v1 environment. In this implementation:

- An episode terminates when:
  - The pole angle θ exceeds $\pm 12°$.
  - The cart position x exceeds ± 2.4.
- The `done` flag is set to `True`.

1.2 Acrobot Markov Decision Process (MDP)

The Acrobot is an another classic reinforcement learning environment and a benchmark problem for studying control systems. It consists of a two-link pendulum (similar to a gymnast swinging on a bar), where the objective is to swing the tip of the second link (the end effector) above a specified height by applying torque to the first joint.

The Acrobot environment can be described as a Markov Decision Process (MDP) using the tuple (S, A, P, R, γ), where:

1.2.1 State Space (S)

The state of the Acrobot system is represented by a six-dimensional continuous vector:

- Cosine and Sine of the angle of the first link relative to the fixed base (measured from the vertical).
- Cosine and Sine of the angle of the second link relative to the first link.
- Angular velocity of the first link.
- Angular velocity of the second link.

These angles are typically constrained as follows:

$$\theta_1, \theta_2 \in [-\pi, \pi]$$

$$\dot{\theta}_1, \dot{\theta}_2 \text{ are unbounded but can be clipped.}$$

The full state space can be written as:

$$S = \{\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\}$$

1.2.2 Action Space (A)

The action space is discrete, with three possible actions representing the torques applied to the joints:

- $a = -1$: Apply a negative torque to the first joint.
- $a = 0$: Apply no torque (coast).
- $a = +1$: Apply a positive torque to the first joint.

1.2.3 Transition Dynamics ($P(s'|s, a)$)

The dynamics of the Acrobot system are governed by the equations of motion derived from the physics of coupled pendulums. These equations calculate the accelerations based on:

- The torques applied.
- Gravitational forces acting on the links.
- The lengths and masses of the links.

These dynamics are deterministic and highly non-linear, making the Acrobot a challenging control problem.

1.2.4 Reward Function ($R(s, a)$)

The reward function is typically sparse:

$$R(s, a) = -1 \quad \text{for every time step to incentivize reaching the goal as quickly as possible.}$$

The goal is to minimize the total cumulative reward over the course of an episode. The episode terminates when the end of the lower link (the hand) reaches a certain height (above a predefined threshold) or after a maximum number of time steps.

1.2.5 Discount Factor (γ)

The discount factor is typically set to a value close to 1 (e.g., $\gamma = 0.99$) to emphasize long-term performance while encouraging the agent to solve the task efficiently.

1.2.6 Initial State

The initial state is randomly generated within a small range around the equilibrium position. Each of the six state variables is initialized within the range $[-0.1, 0.1]$. This randomness ensures the agent learns a robust policy that generalizes to varying starting conditions.

1.2.7 Terminal State

The episode terminates when the following condition is met:

$$-\cos(\theta_1) - \cos(\theta_2 + \theta_1) \geq 1.0$$

This indicates that the second link has been successfully swung above the goal height, marking the completion of the task.

Additionally, the episode terminates if the task is not completed within a fixed number of time steps, such as 500, or if a hard timeout is reached. This can be implemented using the following logic:

$$\texttt{while not done or not timeout or episode\_steps} \geq 2000$$

Here, `done` corresponds to the success condition (the swing above the goal height), `timeout` ensures the episode terminates after a specific time, and `episode_steps` imposes an upper limit to prevent excessively long episodes. This ensures the simulation is efficient and adheres to constraints.

1.2.8 Objective

The objective of the Acrobot MDP is to learn a policy $\pi(s, a)$ that maximizes the expected cumulative reward:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

This translates to minimizing the number of time steps required to swing the end effector above the target height, starting from different initial states.

1.2.9 Simulation Environment

For the project the Acrobot problem is implemented in the OpenAI Gym framework, specifically the `Acrobot-v1` environment. Key features include:

Termination Conditions

- The episode ends when the tip of the second link is above the target height, defined mathematically as:

  $$y = -\cos(\theta_1) - \cos(\theta_2 + \theta_1) \geq 1.0$$

  This condition indicates successful completion of the task.

- The episode also terminates after a fixed number of time steps (e.g., 500 steps) if the goal is not achieved.

- Additionally, the episode ends if a predefined maximum number of time steps is reached, such as 2000, or if a timeout condition occurs due to environmental or computational constraints.

1.3 LunarLander Markov Decision Process (MDP)

The LunarLander-v2 is another popular reinforcement learning environment that challenges an agent to land a lunar module on a designated landing pad while conserving fuel and avoiding crashes. Below is a comprehensive description of the MDP, including the initial and terminal states.

1.4 Components of the MDP

The LunarLander-v2 environment can be described as an MDP using the tuple (S, A, P, R, γ), where:

1.4.1 State Space (S)

The state of the lander is represented as an 8-dimensional continuous vector:

- Horizontal position of the lander (x) relative to the center of the screen.
- Vertical position of the lander (y) relative to the ground.
- Horizontal velocity of the lander ($\dot{x}$).
- Vertical velocity of the lander ($\dot{y}$).
- Angle of the lander (θ) relative to the vertical axis.
- Angular velocity of the lander ($\dot{\theta}$).
- Boolean flag indicating whether the left leg is in contact with the ground:
  - 1: Contact with the ground.
  - 0: No contact.
- Boolean flag indicating whether the right leg is in contact with the ground:
  - 1: Contact with the ground.
  - 0: No contact.

The state space is continuous, with the following constraints:

- (x, y): Typically bounded within the dimensions of the simulation window.
- Velocities $(\dot{x}, \dot{y}, \dot{\theta})$: Technically unbounded but often clipped for practical purposes.
- Angle θ: Limited to $[-\pi, \pi]$.

1.4.2 Action Space (A)

The action space is discrete, with four possible actions:

- $a = 0$: Do nothing.
- $a = 1$: Fire the left orientation engine (tilts the lander clockwise).
- $a = 2$: Fire the right orientation engine (tilts the lander counterclockwise).
- $a = 3$: Fire the main engine (provides upward thrust).

1.4.3 Transition Dynamics ($P(s'|s, a)$)

The dynamics are governed by a 2D physics simulation (using Box2D), incorporating:

- Forces applied by the engines.
- Gravitational pull on the lander.
- Contact forces when the lander interacts with the ground or landing pad.

The next state s' is computed based on the laws of motion, factoring in engine thrust, fuel consumption, and physical constraints.

1.4.4 Reward Function ($R(s, a)$)

The reward structure encourages successful landings:

- $+100$ to $+140$ for landing the module within the designated pad.
- -100 for crashing the module.
- Small positive rewards for reducing the distance to the landing pad.
- Penalties for using fuel (e.g., firing the main engine).
- Additional rewards for stabilizing the lander's angle (θ).

1.4.5 Discount Factor (γ)

The discount factor is typically set to $\gamma = 0.99$, emphasizing long-term success (successful landing) while encouraging efficient fuel use and quick decision-making.

1.4.6 Initial State

The initial state is randomized to provide diverse learning scenarios:

- The lander starts at a random horizontal position (x) and height (y) above the landing pad.
- Initial horizontal and vertical velocities are small random values.
- The initial angle (θ) and angular velocity ($\dot{\theta}$) are small random values.
- Both legs are initially not in contact with the ground ($c_1 = 0$, $c_2 = 0$).

1.4.7 Terminal State

The episode terminates when:

- The lander successfully touches down on the landing pad.
- The lander crashes onto the ground.
- The lander moves far beyond the simulation window bounds.
- A maximum number of time steps is reached (e.g., 500 steps).
- A timeout condition occurs, ensuring the episode ends if it exceeds a global time limit (e.g., 2000 steps across multiple episodes or computations).

The maximum number of time steps per episode and the timeout condition ensure that the simulation runs efficiently and avoids indefinite execution, even if the task is not completed.

1.4.8 Objective

The agent's goal is to learn a policy $\pi(s, a)$ that maximizes the expected cumulative reward:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

This involves minimizing fuel consumption, avoiding crashes, and landing successfully in the shortest time.

1.4.9 Simulation Environment

The LunarLander-v2 environment is implemented in the OpenAI Gym library, with the following features:

- **Physics Engine:** Simulated using Box2D.
- **Landing Pad:** A fixed location in the center of the screen with visual markers to guide the agent.
- **Environment Bounds:** The lander is reset if it moves far beyond the simulation window.

2 Reinforce with Baseline

2.1 Introduction

The Reinforce algorithm is a fundamental policy gradient technique in reinforcement learning that aims to optimize a policy by maximizing the expected cumulative reward over time. While effective, the vanilla Reinforce algorithm often struggles with high variance in its gradient estimates, which can lead to slow and unstable learning. This high variance makes it difficult for the algorithm to reliably update the policy and converge efficiently. To mitigate this issue, an improved version called **Reinforce with Baseline** introduces a baseline function, denoted as $b(s)$, which helps to reduce the variance in the gradient estimates while maintaining their unbiased nature. The baseline acts as a reference

point, allowing the algorithm to more effectively assess the advantages of taking specific actions in different states. By subtracting the baseline from the predicted returns, the algorithm refines its evaluation of how much better or worse an action is compared to the baseline, leading to more stable and accurate updates. This reduced variance results in more reliable policy updates, making learning faster and more efficient. The baseline itself can take several forms: it can be a fixed constant, a random variable, or a learned function, such as the state value function $V(s)$ or an approximation. This flexibility in defining the baseline further enhances the algorithm's ability to converge more quickly and effectively.

In summary, Reinforce with Baseline provides a robust and efficient method for policy optimization by balancing the variance reduction of vanilla Reinforce with the potential bias introduced by value function approximations.

2.2 Policy Gradient Theorem with Baseline

The policy gradient theorem with a baseline is expressed as:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \big(q_\pi(s,a) - b(s)\big) \nabla_\theta \pi(a|s,\theta),$$

where:

- $J(\theta)$: The expected reward under policy π.
- $\mu(s)$: The stationary distribution of states under policy π.
- $q_\pi(s,a)$: The state-action value function under policy π.
- $b(s)$: The baseline function, which does not depend on the action a.
- $\nabla_\theta \pi(a|s,\theta)$: The gradient of the policy with respect to its parameters θ.

Key Property: The baseline $b(s)$ can be any arbitrary function, including a random variable, as long as it is independent of the action a. This ensures that subtracting the baseline does not introduce bias into the gradient estimate because:

$$\sum_a b(s) \nabla_\theta \pi(a|s,\theta) = b(s) \nabla_\theta \bigg(\sum_a \pi(a|s,\theta) \bigg) = b(s) \nabla_\theta 1 = 0.$$

Using the policy gradient theorem, the update rule for Reinforce with Baseline is derived as:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \big(G_t - b(S_t)\big) \frac{\nabla_\theta \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)},$$

where:

- G_t: The return from state S_t under policy π.
- α: The learning rate.

This update rule extends the vanilla Reinforce algorithm and reduces the variance of updates, thereby accelerating the learning process.

2.3 Components

1. **Expected Reward:** $J(\theta)$ represents the average reward obtained over all possible state-action sequences under policy π.
2. **State-Action Value Function:** $q_\pi(s,a)$ is the expected cumulative reward of taking action a in state s and following policy π thereafter.
3. **Baseline Function:** $b(s)$ serves as a reference point for evaluating the advantage of each state-action pair. It can be any function that does not depend on the action a.

2.4 Benefits

1. **Reduced Variance:** By incorporating the baseline, the algorithm achieves a significant reduction in the variance of gradient estimates, leading to more stable learning.
2. **Efficient Learning:** The lower variance enables faster convergence and improves the overall performance of the learning process.
3. **Focus on Improvement:** The baseline ensures that updates focus on improving the policy in states where it performs poorly compared to the baseline.

2.5 Pseudo Code

Reinforce with Baseline Algorithm:

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 1: Algorithm for Reinforce with baseline

1. Initialize policy parameters θ and baseline parameters w.
2. Hyperparameters:
   - Step size for policy parameters (α_θ): Controls the magnitude of parameter updates. A higher step size can lead to faster convergence, but may also cause instability in the learning process.
   - Step size for baseline parameters (α_w): Determines how quickly the baseline adapts. A larger step size may lead to faster convergence of the baseline, but it can be more sensitive to noise in the data.
   - Discount factor (γ): Balances the importance of short-term vs. long-term rewards. A higher discount factor places more importance on future rewards, which can encourage long-term strategies.
   - Eligibility trace decay rate (λ): Governs the decay of historical information. A higher decay rate results in a longer averaging window, potentially making the learning process more stable but slower.
3. For each episode:
   (a) Generate a trajectory of states, actions, and rewards.
   (b) Compute returns G_t for each time step.
   (c) Update the baseline $b(s)$ (if learned).
   (d) Update the policy parameters using:

$$\theta \leftarrow \theta + \alpha_\theta \big(G_t - b(S_t)\big) \nabla_\theta \log \pi(A_t|S_t,\theta).$$

- **Step size tuning:** The step sizes for both the policy parameters and baseline parameters should be carefully tuned to achieve the best performance. A good approach is to start with a small step size and gradually increase it until the desired learning speed and stability are achieved.
- **Discount factor considerations:** The discount factor controls the trade-off between short-term and long-term rewards. A higher value gives more weight to future rewards, which may lead to more long-term, strategic decision-making.

- **Eligibility trace decay rate tuning:** The eligibility trace decay rate determines how quickly the algorithm forgets about past state-action pairs. A higher value makes the algorithm more responsive to recent experiences but may also make it more sensitive to noise in the data.

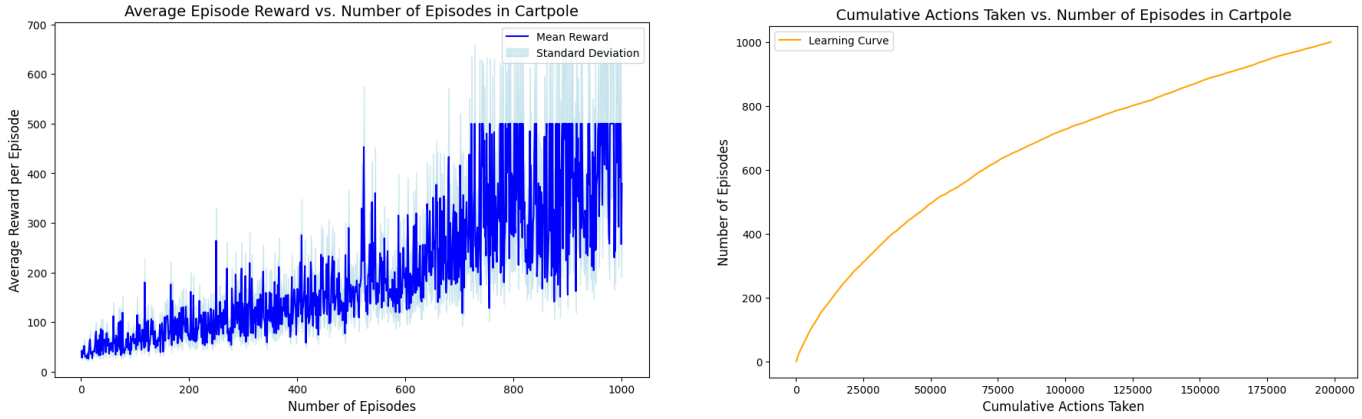
Reinforce with Baseline is a powerful extension of the vanilla Reinforce algorithm that significantly improves learning stability and efficiency in reinforcement learning tasks. By introducing a baseline function to reduce the variance of gradient estimates, it enables faster convergence and more robust policy optimization.

2.6 Experiments & Results : REINFORCE with Baseline

4.6.1 - Cartpole

(A) Best Performing Experiment : In this experiment, we applied the REINFORCE algorithm with a baseline to the Cartpole MDP. The goal is to train an agent to balance a pole on a cart by maximizing the cumulative reward over episodes. The hyperparameters used in this experiment were optimized based on prior exploration and are as follows:

- Discount factor, $\gamma = 0.925$
- Step size for learning the policy approximation function, $\alpha_\theta = 0.01$
- Step size for learning the value approximation function, $\alpha_w = 0.01$
- Number of training episodes, $n = 1000$



(a) Average Episode Reward vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

Figure 2: Comparison of Average Episode Reward and Learning Curve in Cartpole

The graph depicts the average reward per episode achieved by the agent over 1000 episodes, as well as the standard deviation, which provides insight into the stability of the agent's performance.

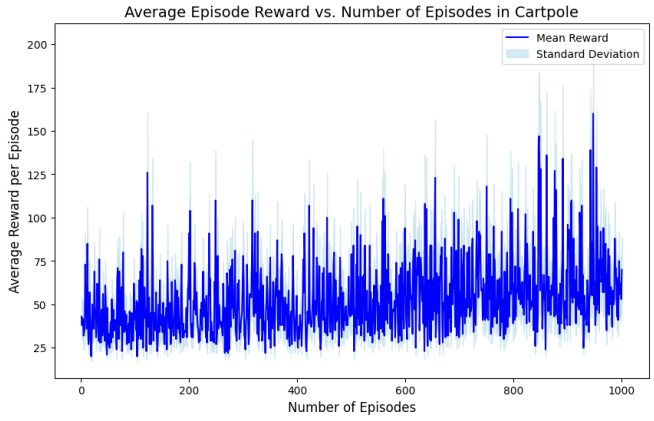
- **X-axis:** The X-axis represents the number of episodes (ranging from 0 to 1000). This shows the progression of the agent's learning over time, with each point corresponding to the agent's performance after a particular number of episodes.
- **Y-axis:** The Y-axis shows the average reward per episode. In the Cartpole task, the agent receives a reward for each timestep the pole is balanced on the cart. The goal is to maximize the average reward over time.
- **Blue Line (Mean Reward):** The blue line represents the mean reward per episode, which steadily increases as training progresses. This indicates that the agent is successfully learning the task of balancing the pole on the cart, achieving higher rewards over time.
- **Light Blue Shaded Region (Standard Deviation):** The shaded region indicates the standard deviation of the rewards, which reflects the variability in the agent's performance. Initially, the agent's performance is highly variable, as seen by the wider shaded region in the early episodes. As training progresses, the standard deviation decreases, indicating that the agent's performance becomes more consistent.

- **Learning Trajectory:** The steady increase in mean reward after the initial phase of exploration indicates that the agent has successfully learned to balance the pole for longer durations. The reduction in standard deviation suggests that the agent's policy has stabilized.

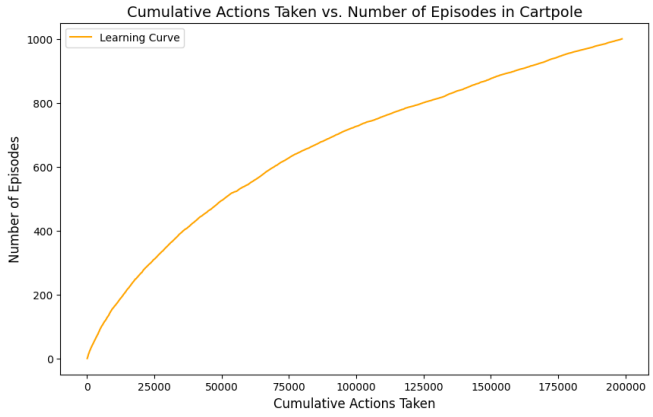
The graph demonstrates the performance of an agent trained using the REINFORCE algorithm with a baseline in the Cartpole environment over 1000 episodes. Initially, the agent's performance is unstable, with the average reward being lower and the standard deviation relatively high, reflecting the exploration phase of learning. However, as training progresses, both the mean reward increases and the standard deviation decreases, showing that the agent is improving its policy and becoming more consistent in its actions. This steady rise in reward, alongside the reduction in variance, indicates that the agent is successfully learning and stabilizing its performance over time. The choice of parameters, including the discount factor ($\gamma = 0.925$) and learning rates for both the policy and value function ($\alpha_\theta = \alpha_w = 0.01$), appears to be optimal for achieving the best learning outcomes.

Other Results

(B) In contrast, we also conducted an experiment where the step sizes α_θ and α_w were set to excessively small values, $\alpha_\theta = \alpha_w = 0.001$ and also $\alpha_\theta = \alpha_w = 0.0001$, while keeping the discount factor ($\gamma = 0.925$) constant. The graph represents the results obtained with smaller step sizes. The reward increases very slowly, and the agent's performance shows signs of stagnation as it leads to extremely small updates to the policy and value function, causing very slow learning and poor performance. The small step sizes limit the agent's ability to explore the state space effectively. As a result, the agent struggles to discover optimal policies, and learning stagnates for a significant portion of the training. The unfavorable results observed in this experiment can be attributed to the excessively small step sizes used for updating the policy and value approximation functions. These tiny step sizes result in slow convergence and poor learning performance. To improve performance, it is recommended to experiment with larger step sizes, such as $\alpha_\theta = \alpha_w = 0.01$ or $\alpha_\theta = \alpha_w = 0.015$, to strike a better balance between convergence speed and stability in the learning process.

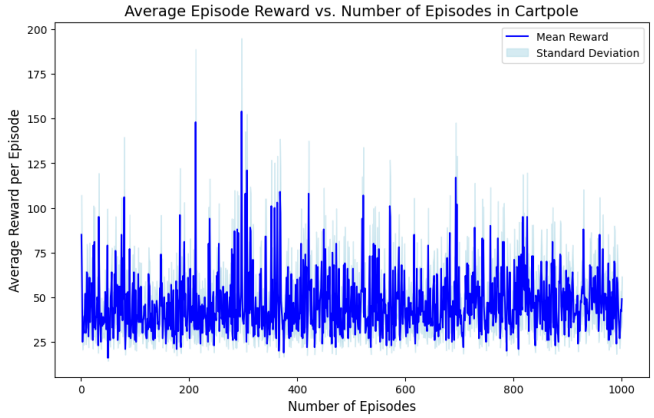


(a) Average Episode Reward vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$)

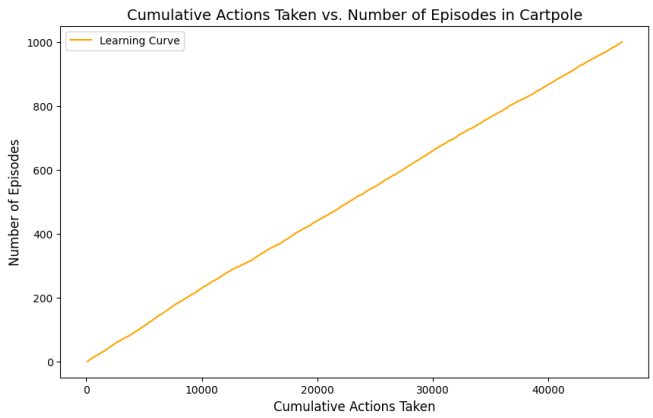


(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$)

Figure 3: Comparison of Average Episode Reward and Learning Curve in Cartpole



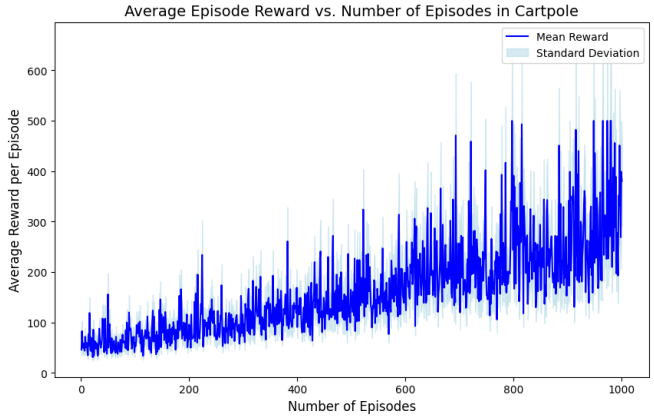
(a) Average Episode Reward vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$)



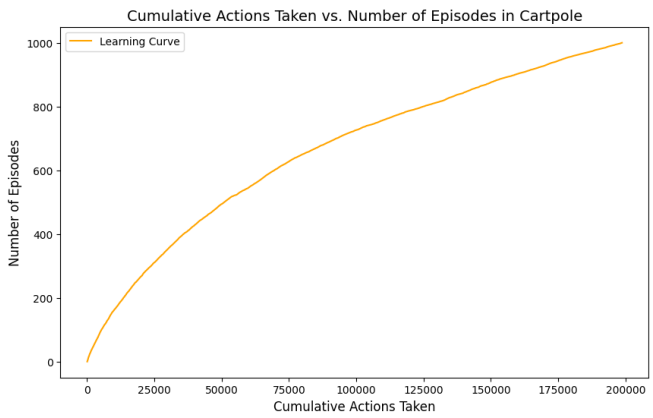
(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$)

Figure 4: Comparison of Average Episode Reward and Learning Curve in Cartpole

(C) When compared to results using very small step sizes ($\alpha_\theta = 0.001$ and $\alpha_w = 0.001$), and even smaller ones ($\alpha_\theta = 0.0001$ and $\alpha_w = 0.0001$), using slightly larger step sizes that performed the best ($\alpha_\theta = 0.01$ and $\alpha_w = 0.01$) along with an increased discount factor ($\gamma = 0.99$) yielded better results. The larger step sizes allowed the agent to learn faster, improving the efficiency of the learning process. This setup strikes a good balance between exploring different actions and making steady progress. Furthermore, the higher discount factor of $\gamma = 0.99$ caused the agent to place greater importance on future rewards, which influenced its decision-making accordingly. Although this experiment did not perform as well as the one with step sizes of $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, and $\gamma = 0.925$, it reached a reward of 500 more quickly, indicating that it was learning more efficiently in time. The experiment with $\gamma = 0.925$ showed a better overall performance, with higher rewards per episode. Near 800 episodes, the rewards fluctuated, but remained consistently near the 500 mark for the average reward per episode. In contrast, the $\gamma = 0.99$ set-up demonstrated more volatility, with significant spikes in reward reaching 500, but with fluctuations across consecutive episodes. This suggests that while a higher discount factor may lead to faster convergence, a slightly lower γ (such as $\gamma = 0.925$) could offer better long-term performance, as seen in the experiment with $\gamma = 0.925$.



(a) Average Episode Reward vs. Number of Episodes in Cartpole ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

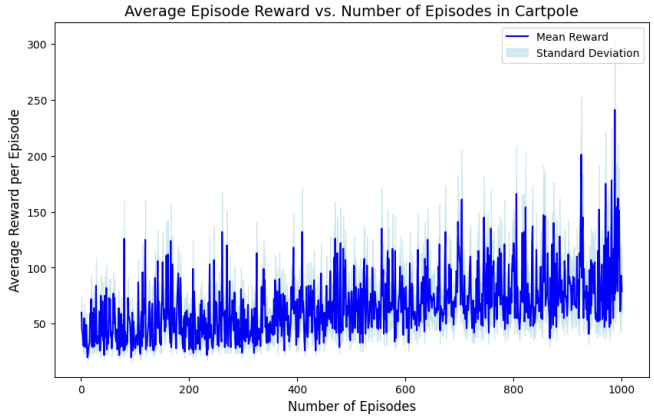


(b) Learning Curve in Cartpole ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

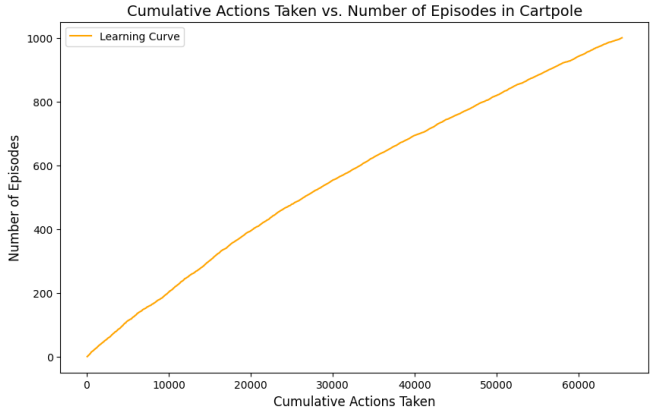
Figure 5: Comparison of Average Episode Reward and Learning Curve in Cartpole

(D) We also tested this experiment, with step sizes of $\alpha_\theta = 0.01$ and $\alpha_w = 0.01$, and a low discount factor $\gamma = 0.80$, the agent demonstrates steady learning over time, with a gradual increase in rewards per episode and moderate fluctuations. The smaller step sizes result in more stable updates, allowing the agent to explore the environment more carefully. As the agent learns, the rewards increase consistently, but the learning process is slower compared to scenarios with larger step sizes. The discount factor $\gamma = 0.80$ means that the agent places less value on future rewards, which contributes to a more gradual improvement in performance. This setup offers a balanced approach between stability and efficiency, showing steady progress without the instability seen in earlier experiments with larger step sizes. When comparing this

setup with a previous experiment in which $\gamma = 0.925$, we observe a noticeable difference in performance. With $\gamma = 0.80$, the agent values future rewards less, leading to lower overall rewards per episode. The graph shows a steady but slower increase in average reward, and although it improves with time, it does not reach the same level of performance as the experiment with $\gamma = 0.925$. In contrast, with $\gamma = 0.925$, the agent places greater emphasis on future rewards, resulting in faster and greater improvements in rewards per episode. This leads to a more efficient learning process, reflected in faster and higher peaks on the graph. The experiment with $\gamma = 0.80$ tends to be slower and yields lower rewards, suggesting that a higher gamma can be more beneficial for faster convergence and better overall performance.



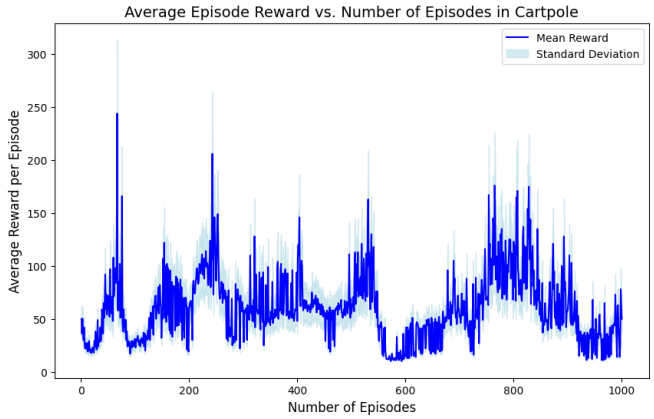
(a) Average Episode Reward vs. Number of Episodes in Cartpole ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)



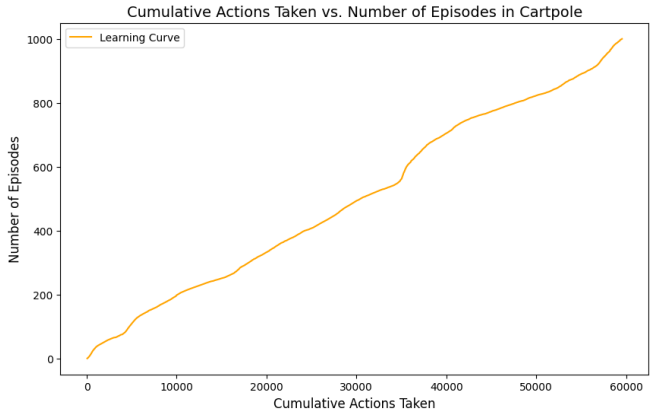
(b) Learning Curve in Cartpole ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

Figure 6: Comparison of Average Episode Reward and Learning Curve in Cartpole.

(E) We also tested this using a larger step sizes of $\alpha_\theta = 0.3$ and $\alpha_w = 0.3$, along with a discount factor (γ) of 0.925, produced a graph that shows greater fluctuations compared to the previous experiments. The larger step sizes caused faster learning at first, but also led to more instability, as seen in the larger variations in the rewards. This indicates that the agent was updating its policy too quickly, which resulted in oscillations and inconsistent performance. The inclusion of a discount factor of $\gamma = 0.925$ still helped the agent consider future rewards, but the balance between speed and stability was not optimal. Although the agent learned faster initially, the high step sizes may have caused it to miss more stable and effective learning paths. This suggests that further tuning of step sizes and the discount factor may be necessary to find the right balance between fast learning and stable progress.



(a) Average Episode Reward vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.3$, $\alpha_w = 0.3$)



(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.3$, $\alpha_w = 0.3$)

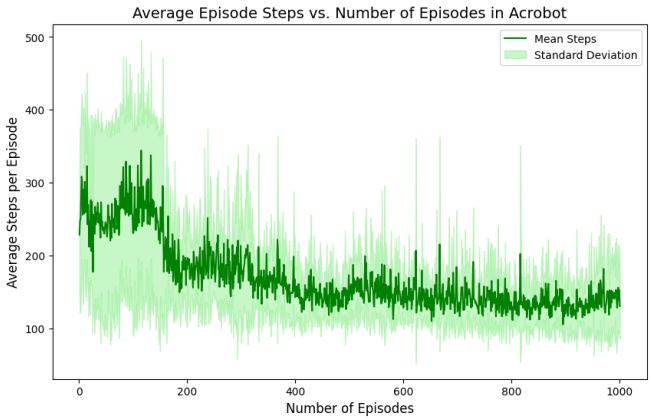
Figure 7: Comparison of Average Episode Reward and Learning Curve in Cartpole

4.6.2 - Acrobot

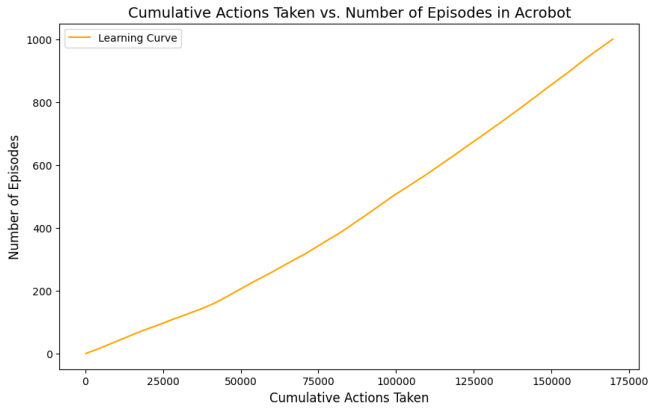
(A) Best Performing Experiment: In this experiment, we applied the REINFORCE algorithm with a baseline to the Acrobot MDP. The goal is to train an agent to swing up and balance a two-link underactuated pendulum, known as

the Acrobot, by maximizing the cumulative reward over episodes. The hyperparameters used in this experiment were optimized based on prior exploration and are as follows:

- Discount factor, $\gamma = 0.925$
- Step size for learning the policy approximation function, $\alpha_\theta = 0.01$
- Step size for learning the value approximation function, $\alpha_w = 0.01$
- Number of training episodes, $n = 1000$



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)



(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

Figure 8: Comparison of Average Episode Steps and Learning Curve in Acrobot

The graph depicts the average number of steps per episode achieved by the agent over 1000 episodes, as well as the standard deviation, which provides insight into the stability of the agent's performance.

- **X-axis:** The X-axis represents the number of episodes (ranging from 0 to 1000). This shows the progression of the agent's learning over time, with each point corresponding to the agent's performance after a particular number of episodes. Copy
- **Y-axis:** The Y-axis shows the average number of steps per episode. In the Acrobot task, the agent receives a negative reward for each timestep, and the goal is to minimize the number of steps required to swing up and balance the pendulum.
- **Green Line (Mean Steps):** The green line represents the mean number of steps per episode, which decreases as training progresses. This indicates that the agent is successfully learning the task of swinging up and balancing the Acrobot, achieving shorter episodes over time.
- **Green Shaded Region (Standard Deviation):** The shaded region indicates the standard deviation of the steps, which reflects the variability in the agent's performance. Initially, the agent's performance is highly variable, as seen by the wider shaded region in the early episodes. As training progresses, the standard deviation decreases, indicating that the agent's performance becomes more consistent.
- **Learning Trajectory:** The steady decrease in mean steps after the initial phase of exploration indicates that the agent has successfully learned to swing up and balance the Acrobot in fewer steps. The reduction in standard deviation suggests that the agent's policy has stabilized.

The graph demonstrates the performance of an agent trained using the REINFORCE algorithm with a baseline in the Acrobot environment over 1000 episodes. Initially, the agent's performance is unstable, with the average number of steps being higher and the standard deviation relatively high, reflecting the exploration phase of learning. However, as training progresses, both the mean steps decrease and the standard deviation decreases, showing that the agent is improving its policy and becoming more consistent in its actions. This steady decrease in steps, alongside the reduction in variance, indicates that the agent is successfully learning and stabilizing its performance over time. The choice of parameters, including the discount factor ($\gamma = 0.925$) and learning rates for both the policy and value function ($\alpha_\theta = \alpha_w = 0.01$), appears to be optimal for achieving the best learning outcomes in the Acrobot environment.

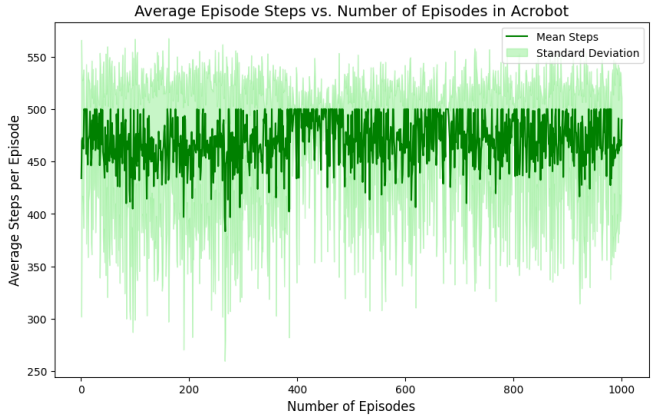
Other Results

(B) In our experiments, we tested different learning rates with the same discount factor of $\gamma = 0.925$: $\alpha_\theta = \alpha_w = 0.01$, $\alpha_\theta = \alpha_w = 0.001$, and $\alpha_\theta = \alpha_w = 0.0001$, to understand their effect on solving the Acrobot task. The goal was to evaluate how different learning rates impact the agent's performance, learning speed, and stability over the course of training.

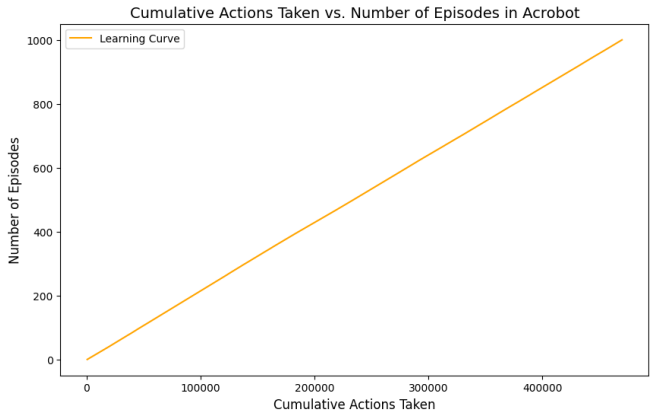
The configuration with higher learning rates of $\alpha_\theta = \alpha_w = 0.01$ consistently outperformed the others. The agent showed steady and efficient learning, with a clear decline in mean steps per episode and a narrowing of the standard deviation over 1000 training episodes. This indicates effective optimization of the agent's policy, where the higher learning rate allowed the agent to adapt more quickly and converge to a solution faster. The relatively high discount factor of 0.925 encouraged the agent to focus on long-term rewards, leading to stable learning throughout the process. Overall, this configuration provided the best balance between fast convergence and reliable learning, making it the most effective approach.

In contrast, when we tested smaller learning rates, such as $\alpha_\theta = \alpha_w = 0.001$, the agent's performance was slower and less stable. With this configuration, the mean steps fluctuated significantly, ranging between 500 and 420, and the agent experienced frequent spikes during the training episodes. While the standard deviation remained moderate at ± 70, the fluctuations indicated that the learning rate was too small to allow for consistent refinement of the policy. This instability suggests that smaller learning rates hinder the agent's ability to explore and learn effectively from the state-action space, leading to slower and less efficient learning.

Furthermore, the performance worsened with even smaller learning rates of $\alpha_\theta = \alpha_w = 0.0001$. In this case, the mean steps stabilized near 420, but the standard deviation increased significantly, reaching around ± 150. This greater variability further slowed the learning process and made the agent's progress even more erratic. The smaller learning rates in both of these configurations limited the agent's ability to adapt quickly, resulting in slower convergence and ultimately suboptimal performance compared to the $\alpha_\theta = \alpha_w = 0.01$ configuration.

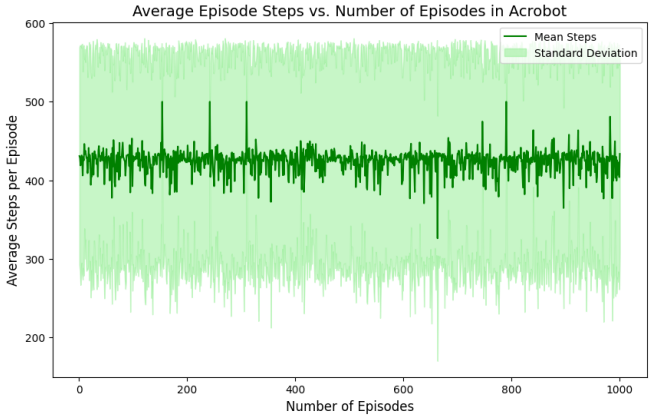


(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$)

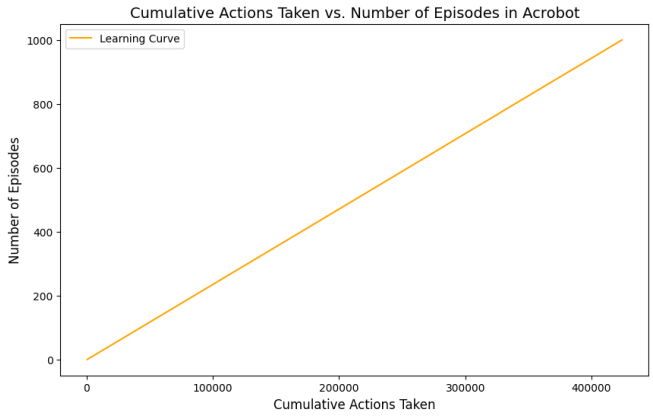


(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$)

Figure 9: Comparison of Average Episode Steps and Learning Curve in Acrobot



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$)



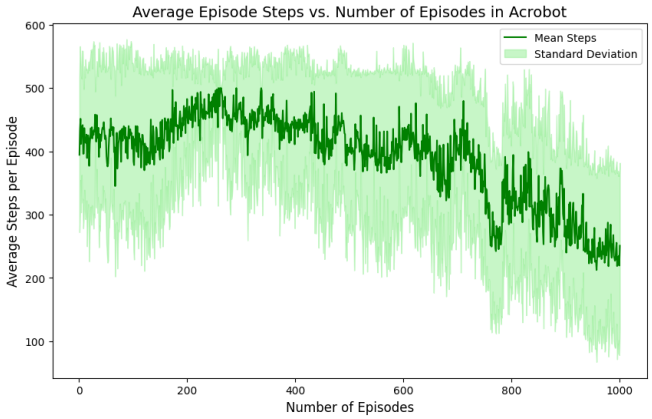
(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$)

Figure 10: Comparison of Average Episode Steps and Learning Curve in Acrobot

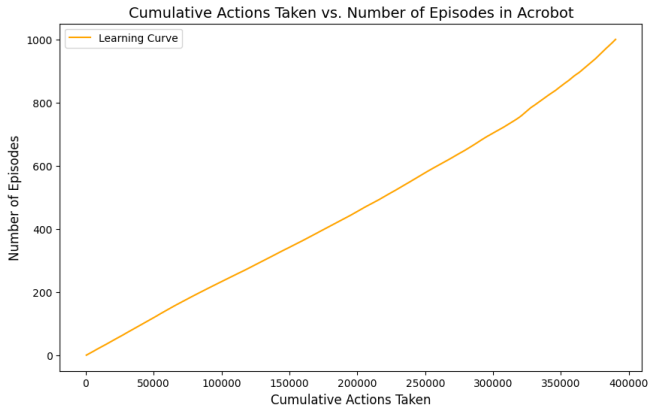
(C) We also tested the configuration with $\gamma = 0.99$, $\alpha_\theta = 0.01$, and $\alpha_w = 0.01$. By increasing the discount factor to $\gamma = 0.99$, we aimed to observe how the agent would prioritize long-term rewards over short-term gains. The graph for this configuration shows a steady decline in the mean number of steps per episode. Initially, the agent takes more steps due to exploration, but as training progresses, the number of steps stabilizes, and the agent's performance becomes more consistent. The green shaded area, representing the standard deviation, narrows slightly over time, reflecting improved stability in the agent's performance.

In contrast, the configuration with $\gamma = 0.925$, $\alpha_\theta = 0.01$, and $\alpha_w = 0.01$ emphasizes short-term rewards more strongly, as the lower discount factor causes the agent to value immediate rewards over future ones. This setup shows faster learning with a sharper decline in mean steps per episode and earlier stabilization of the policy, as indicated by the more rapid reduction of the shaded area. With $\gamma = 0.925$, the agent converges to an effective policy quicker, while still achieving comparable long-term performance.

Overall, the $\gamma = 0.925$ configuration proved to be more efficient, enabling quicker convergence and faster stabilization of the policy. While $\gamma = 0.99$ focused on long-term reward optimization, the $\gamma = 0.925$ setup provided a better balance for faster learning in the Acrobot task.



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)



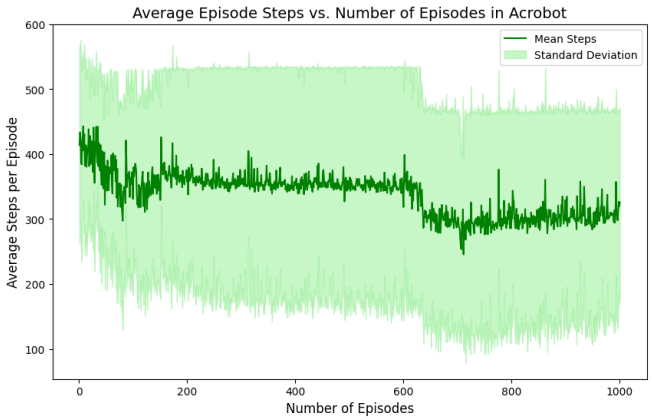
(b) Learning Curve in Acrobot ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

Figure 11: Comparison of Average Episode Steps and Learning Curve in Acrobot.

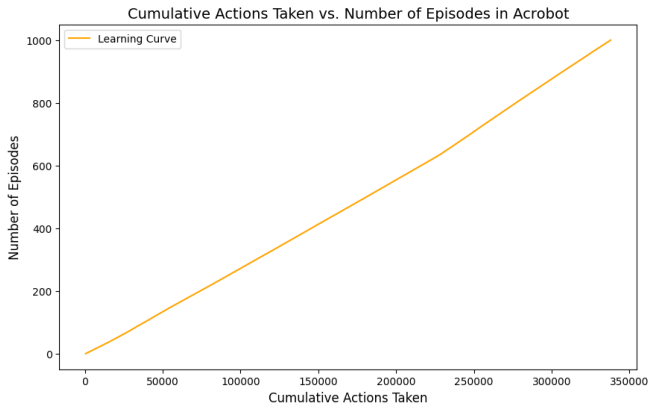
(D) We also tested the configuration with $\gamma = 0.80$, $\alpha_\theta = 0.01$, and $\alpha_w = 0.01$ to observe the effect of a lower discount factor and higher learning rates on the agent's performance. The graph shows a significantly slower decline in the mean steps per episode, starting around 400 and gradually decreasing to approximately 300 over 1000 episodes. This slower progress indicates that the agent struggles to effectively learn the Acrobot task under this configuration. The

green shaded area, representing the standard deviation, remains wide throughout training, often exceeding ±130, which highlights high variability and inconsistent performance.

The lower discount factor ($\gamma = 0.80$) causes the agent to prioritize short-term rewards over long-term objectives, limiting its ability to plan and optimize actions for the ultimate goal of balancing the Acrobot. Additionally, the larger learning rates ($\alpha_\theta = 0.01$ and $\alpha_w = 0.01$) introduce instability in the policy updates, preventing the agent from converging to an effective solution. This combination of a reduced focus on long-term rewards and larger learning rates leads to slow improvement and high variability in the agent's performance. Hence, this configuration resulted in slower progress and instability, making it less suitable for solving the Acrobot task effectively compared to configurations with higher discount factors.



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

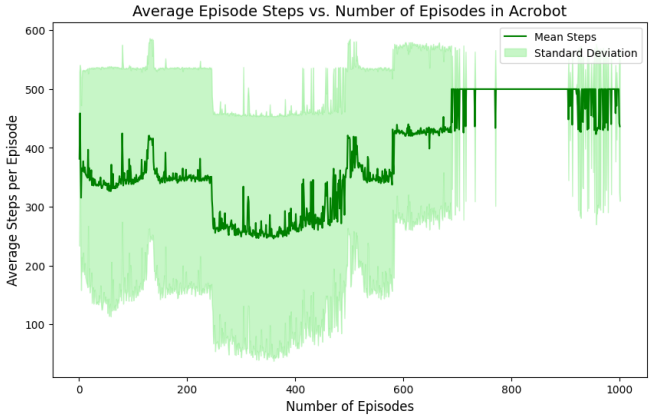


(b) Learning Curve in Acrobot ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

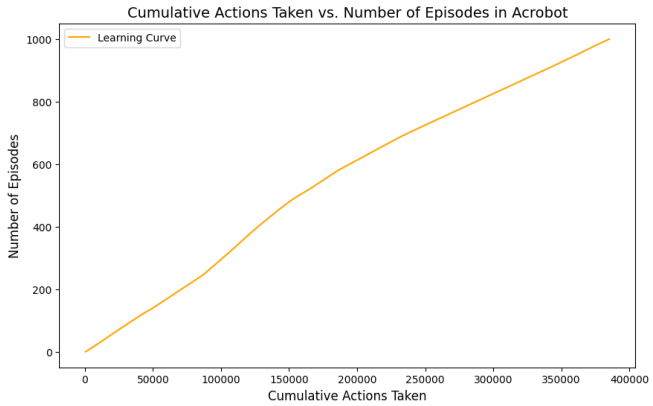
Figure 12: Comparison of Average Episode Steps and Learning Curve in Acrobot.

(E) We also tested the configuration with $\gamma = 0.925$, $\alpha_\theta = 0.3$, and $\alpha_w = 0.3$ to examine the effect of higher learning rates on the agent's ability to learn and maintain an effective policy. The graph for this configuration exhibits an unusual learning trajectory. The mean steps per episode start around 400-350 and initially decline to approximately 300 by episode 250, suggesting early progress in learning the Acrobot task. However, after episode 450, the mean steps begin to increase steadily, reaching around 500 by episode 750, where they plateau. This indicates a failure to maintain the learned policy, with the agent regressing in performance over time.

The standard deviation remains high (approximately ±130) during the first 750 episodes, reflecting significant variability and instability in the agent's performance. After episode 750, the variability decreases notably, indicating more consistent, albeit suboptimal, performance. Near the end of training (around 900-950 episodes), the standard deviation increases slightly, showing minor instability, though not to the extent observed earlier. The high learning rates ($\alpha_\theta = 0.3$ and $\alpha_w = 0.3$) likely caused instability in updates, leading to overcorrections and preventing the agent from converging to an effective policy. While the discount factor ($\gamma = 0.925$) is generally favorable for balancing long- and short-term rewards, the excessive learning rates hinder the agent's ability to refine its policy effectively. This configuration highlights the trade-off between fast learning and stability, with overly aggressive updates leading to poor long-term performance.



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.3$, $\alpha_w = 0.3$)



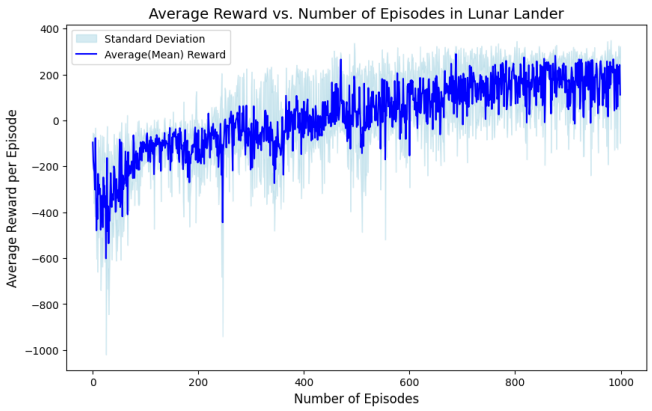
(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.3$, $\alpha_w = 0.3$)

Figure 13: Comparison of Average Episode Steps and Learning Curve in Acrobot.

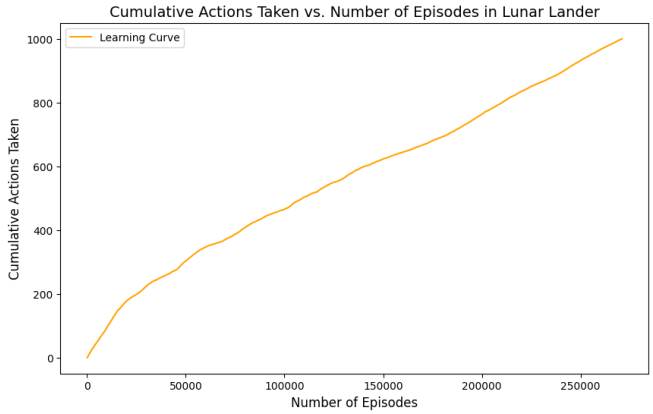
4.6.3 - Lunar Lander

(A) Best Performing Experiment: In this experiment, we applied the REINFORCE algorithm with a baseline to the Lunar Lander environment. The objective was to train an agent to land a spacecraft smoothly on a designated landing pad, maximizing the cumulative reward over 1000 episodes. The hyperparameters used in this experiment were fine-tuned through prior exploration and are as follows:

- Discount factor, $\gamma = 0.99$
- Step size for learning the policy approximation function, $\alpha_\theta = 0.01$
- Step size for learning the value approximation function, $\alpha_w = 0.01$
- Number of training episodes, $n = 1000$



(a) Average Episode Steps vs. Number of Episodes in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)



(b) Learning Curve in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

Figure 14: Comparison of Average Episode Steps and Learning Curve in Lunar Lander.

The graph depicts the average reward per episode achieved by the agent over 1000 episodes, with the inclusion of standard deviation to provide insight into the variability of the agent's performance.

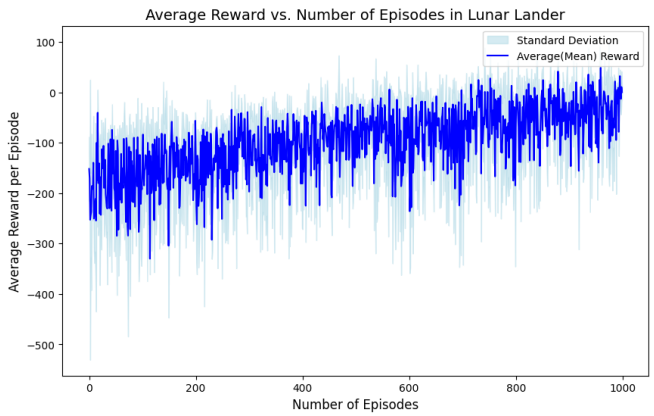
- **X-axis:** The X-axis represents the number of episodes (ranging from 0 to 1000). Each point shows the agent's performance after a specific number of training episodes.

- **Y-axis:** The Y-axis represents the average reward per episode. A higher reward corresponds to a better performance in the Lunar Lander task, which involves safely landing the spacecraft while minimizing fuel usage and avoiding crashes.
- **Blue Line (Mean Reward):** A steadily increasing mean reward demonstrates consistent performance improvement. By episode 900, the agent achieves an average reward above +200, indicating mastery of the landing task.
- **Blue Shaded Region (Standard Deviation):** The shaded region indicates the standard deviation of the rewards, reflecting the variability in the agent's performance. Initially, the standard deviation is high, showing significant variability during the exploration phase. Over time, the standard deviation decreases slightly, indicating more consistent performance by the agent.

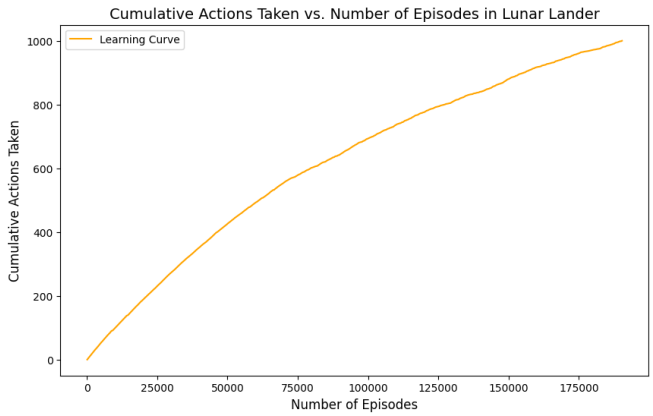
The agent shows rapid improvement within the first 200 episodes, transitioning from highly negative average rewards (e.g., crashes) to rewards closer to zero. The performance even stabilizes more between episodes 700–1000, with the mean reward exceeding around 200, indicating the agent has learned an effective policy for landing. The narrowing standard deviation suggests improved reliability, although some fluctuations persist due to stochastic action selection and environment randomness. This graph demonstrates the effectiveness of the REINFORCE algorithm with a baseline in the Lunar Lander environment. The steady increase in average rewards and reduction in standard deviation illustrate the agent's ability to learn and stabilize its landing policy over time. The chosen hyperparameters ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$) optimize the balance between exploration and exploitation, achieving robust performance by episode 900. These results confirm that this setup achieves the best learning outcomes in this environment compared to other configurations.

Other Results

(B) We also tested the configuration with $\gamma = 0.99$, $\alpha_\theta = 0.001$, and $\alpha_w = 0.001$, which demonstrates steady progress in optimizing the agent's policy. The graph reveals an upward trend in the average reward per episode during training, indicating the agent's ability to prioritize long-term rewards. However, the rewards plateau around 0 by the end of 1000 episodes, which is significantly lower compared to the best-performing configuration, where rewards consistently reach around +200 within the same timeframe. In the initial episodes, this setup exhibits significant fluctuations, with rewards occasionally dropping below -100 and even -200, as highlighted by the wider standard deviation (light blue shaded area). These fluctuations indicate instability in learning, likely caused by the higher learning rate ($\alpha_\theta = 0.001$), which accelerates updates but introduces excessive volatility. Over time, the standard deviation narrows, reflecting improved stability, but the agent fails to optimize its policy adequately, resulting in rewards stagnating near zero. Compared to the best-performing configuration, which achieves smooth and stable progress early on, this setup struggles with high variability in the initial episodes and converges to suboptimal rewards. The larger standard deviation and slower stabilization further highlight the agent's difficulty in learning a consistent policy, unlike the best-performing case, where variability is minimized early, and rewards steadily increase to a much higher range.



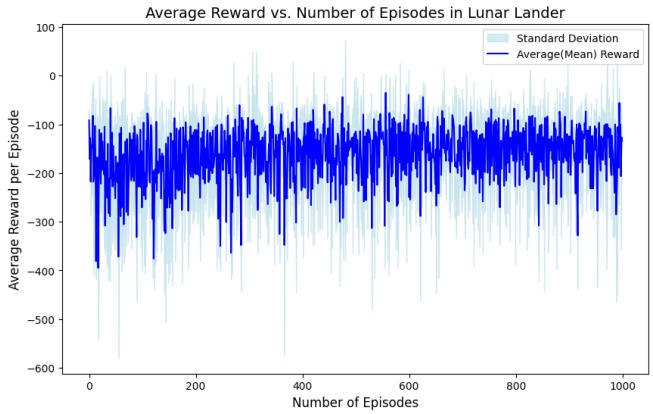
(a) Average Episode Steps vs. Number of Episodes in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$)



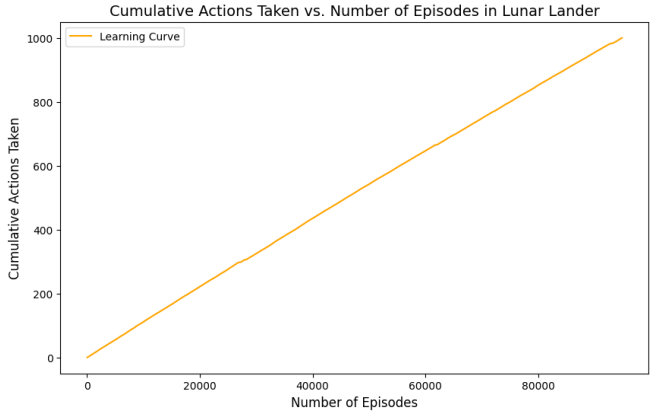
(b) Learning Curve in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$)

Figure 15: Comparison of Average Episode Steps and Learning Curve in Lunar Lander.

(C) We also tested the configuration with $\gamma = 0.99$, $\alpha_\theta = 0.0001$, and $\alpha_w = 0.0001$, which demonstrates a slower and smoother learning curve. The average reward per episode fluctuates significantly, remaining in the range of -400 to -200 for the majority of the training, and eventually settles around -400 by the end of 1000 episodes. While the smaller learning rate reduces the risk of overshooting, it leads to an extremely slow pace of learning. This delayed convergence prevents the agent from achieving meaningful improvements within the given timeframe. In contrast to the best-performing configuration, which consistently reaches rewards around +200 by 1000 episodes, this setup struggles to optimize the policy effectively. The agent's inability to even reach positive rewards highlights the inefficiency of this configuration. The wider standard deviation throughout training further emphasizes that the agent is stuck in a prolonged exploratory phase, failing to consistently improve its decision-making process.



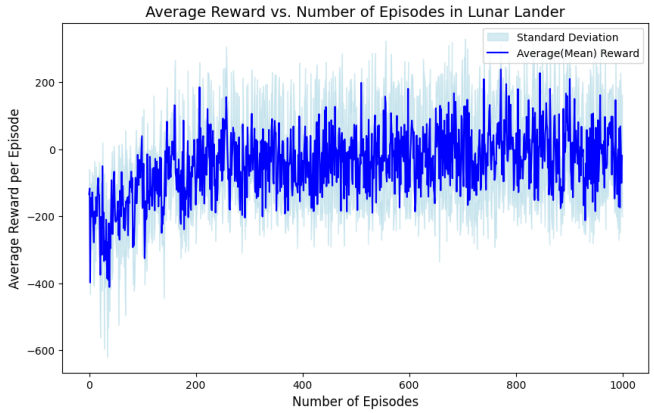
(a) Average Episode Steps vs. Number of Episodes in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$)



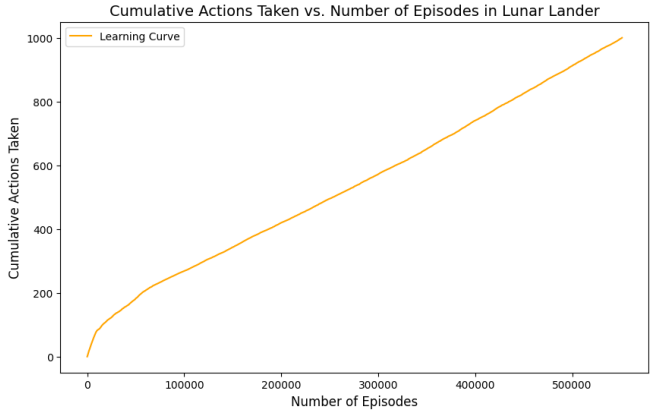
(b) Learning Curve in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$)

Figure 16: Comparison of Average Episode Steps and Learning Curve in Lunar Lander.

(D) We also tested the configuration with a slightly lower discount factor $\gamma = 0.80$, and $\alpha_\theta = \alpha_w = 0.01$, which exhibits a significant amount of fluctuation throughout the training process. The agent reaches an average reward of 0 relatively quickly, within the first 200 episodes, but fails to improve substantially beyond that point. Unlike the best-performing configuration, where the reward steadily increases to approximately +200, the rewards in this setup oscillate frequently between -100 and 100. This persistent oscillation indicates that the agent struggles to stabilize its policy. The high learning rate ($\alpha_\theta = 0.01$) contributes to the volatility, as it causes the updates to overshoot the optimal solutions, preventing consistent improvements. By the end of 1000 episodes, the rewards remain near 0, showcasing the inability of this configuration to effectively optimize the agent's decision-making. In comparison to the best-performing configuration, which exhibits smooth and steady progress with minimal variability, this setup demonstrates excessive instability and fails to achieve meaningful improvements in rewards. The inability to maintain upward progress and the constant oscillations highlight the limitations of using such a high learning rate, particularly in combination with a lower discount factor ($\gamma = 0.80$), which may undervalue long-term rewards.



(a) Average Episode Steps vs. Number of Episodes in Lunar Lander ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)



(b) Learning Curve in Lunar Lander ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$)

Figure 17: Comparison of Average Episode Steps and Learning Curve in Lunar Lander.

3 One-Step Actor-Critic Method

3.1 Introduction

The **One-Step Actor-Critic Method** combines the strengths of policy-based and value-based reinforcement learning by employing an *actor-critic architecture*. This approach overcomes some of the limitations of REINFORCE with baseline algorithms. In REINFORCE algorithm, the baseline (state-value function) is estimated for the first state of each transition to provide a reference for policy updates. However, this baseline is computed before the action is taken and does not account for the subsequent state.

Actor-critic methods improve upon this by using the *state-value function* to evaluate the second state of each transition, thereby providing a more immediate and action-relevant update. In this framework:

- The **actor** refers to the policy, which decides the action to take based on the current state.
- The **critic** refers to the learned value function, which critiques the action by estimating the value of the resulting state.

The state-value function in actor-critic methods is typically learned using *semi-gradient TD(0)*, making it well-suited for online and incremental updates. This integration of policy optimization (*actor*) and temporal difference learning (*critic*) enables faster and more stable convergence compared to pure policy gradient methods.

In policy gradient methods like the actor-critic, actions are selected using a **softmax policy** for smooth transitions towards a deterministic policy, or ϵ-greedy methods to maintain exploration. Hyperparameters such as step sizes, discount factors, and exploration rates are carefully tuned to optimize performance and convergence rates.

3.2 Policy Gradient Equation:

The policy gradient equations remain the same as those of REINFORCE with baseline algorithms. The policy update equation is given by:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta \nabla_\theta \log \pi(A_t|S_t, \theta_t)$$

3.3 Methods:

1. **Fourier Features Generation:**
   `FourierFeatures(nn.Module)` implements a Fourier basis feature representation for the state space.
   (a) **Key Methods:**
      i. `__init__(self, num_dims, num_orders):`
      Initializes the Fourier coefficients, weights, and biases for each dimension of the state space. Each dimension has `num_orders` coefficients, stored as trainable parameters.

ii. `normalize_state(self, state):`
Scales each state variable to a normalized range, based on pre-defined limits. Cosine features use [0, 1] scaling. Sin features use [-1, 1] scaling.

iii. `forward(self, x):`
Computes the Fourier features for the given input state using cosine functions.
Output: A stacked tensor of Fourier-transformed state features.

(b) **Fourier Features and Weights Generation:**
For order-k with d state variables, this generates $(k+1)^d$ features. The Fourier features are computed for each state variable, producing a higher-dimensional feature representation that captures the different frequencies of the state space. Additionally, weights are generated for both the state-value function and the policy network of size $(k+1)^d$.

2. **Policy Network:**
`PolicyNetwork(nn.Module)` maps input features to action probabilities.

(a) **Key Layers:**

i. Three fully connected layers (`fc1`, `fc2`, `fc3`) with ReLU activations.

ii. Output layer uses softmax activation to generate a probability distribution over actions.

(b) **Output:** Action probabilities for the given state.

3. **State-Value Network:**
`StateValueNetwork(nn.Module)` approximates the state-value function $V(s)$.

(a) **Key Layers:**

i. Three fully connected layers (`fc1`, `fc2`, `fc3`) with ReLU activations.

(b) **Output:** A scalar value representing the state value for the given input.

(c) **Parameterized State-Value Function:** The state-value function at any given timestep is computed as:

$$v_w(s) = w^T \phi(s) = \sum_{i=1}^{d} w_i \phi_i(s)$$

where:

- $v_w(s)$ is the state-value function at state s,
- w_i are the weights associated with the features $\phi_i(s)$,
- $\phi(s)$ is the feature vector of the state s,
- d is the number of features in the state representation.

(d) **Feature and Weight Generation:**
The features used in the state-value function are generated using the Fourier transformation process. For order-k with d state variables, the features include $(k+1)^d$ components. The weights for both the state and policy networks are of size $(k+1)^d$.

4. **Actor-Critic Architecture:**
`ActorCritic(nn.Module)` combines policy and state-value networks using Fourier-transformed features.

(a) **Key Components:**

i. `fourier_features:` Fourier basis feature extractor.

ii. `actor_policy_network:` Predicts action probabilities.

iii. `actor_state_value_network:` Predicts state value.

(b) **Key Methods:**

i. `normalize_state(self, state):`
Normalizes the state variables for feature extraction.

ii. `forward(self, x):`
Extracts Fourier features and passes them to the policy and state-value networks.
Output: Action probabilities and state value.

5. **Action Selection:**
   `action_selection(state, actor_critic_model)` selects an action based on the policy network's output.
   (a) **Steps:**
      i. Normalizes the input state and computes action probabilities.
      ii. Samples an action from the categorical distribution generated by the policy network.
   (b) **Output:**
      i. Selected action.
      ii. Log probability of the selected action.
      iii. Action probabilities.
6. **Actor-Critic Training:**
   `actor_critic_function(policy_net, state_value_net, policy_optimizer, state_value_optimizer, env, model, num_episodes, gamma)` trains the actor-critic model using temporal difference (TD) learning.
   (a) **Key Steps:**
      i. Runs episodes to collect state transitions, rewards, and actions.
      ii. Computes the temporal difference error δ for each step:
      $$\delta = r + \gamma \cdot V(s') - V(s)$$
      iii. Updates:
         A. **Policy Network:** Using the actor loss:
         $$\text{Actor Loss} = -\log(\pi(a \mid s)) \cdot \delta$$
         B. **State-Value Network:** Using the critic loss:
         $$\text{Critic Loss} = \delta^2$$
      iv. Logs rewards every 100 episodes.
7. **Environment Interaction:**
   Interaction:
   (a) Resets the environment at the start of each episode.
   (b) Selects an action, observes the next state and reward.
   (c) Applies policy and value updates iteratively.
8. **Action Selection:**
   In policy gradient methods:
   - **Softmax:** Softmax is used for action preference, as it approximates a deterministic policy while converging gradually.
   - **ϵ-Greedy:** Maintains randomness with a probability ϵ for exploring random actions.
9. **Hyperparameter Tuning:**
   Adjust hyperparameters like step sizes (α_θ, α_w), exploration rate (σ), and feature order (n) to optimize the learning and convergence rate. The expected average performance is achieved by iteratively tuning these parameters.

3.4 Benefits

1. **Reduced Variance:** The critic (value function) reduces the variance of the policy gradient by providing a more accurate estimate of the advantage function, leading to more stable updates.
2. **Improved Convergence Speed:** The reduced variance enables faster convergence compared to pure policy gradient methods like REINFORCE, allowing the algorithm to learn more efficiently.
3. **Continuous Improvement:** By using a single-step temporal difference (TD) error, the updates focus on improving the policy locally in a step-by-step manner, which balances exploration and exploitation effectively.
4. **Scalability to Complex Tasks:** The one-step update simplifies the computational overhead compared to multi-step methods, making it suitable for tasks with a continuous state or action space.
5. **Adaptability to Dynamic Environments:** The online nature of the actor-critic algorithm allows it to adapt quickly to changes in the environment by continuously learning from the latest interactions.

3.5 Pseudo Code:

One-step Actor–Critic Algorithm:

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Figure 18: Algorithm for One-step Actor–Critic

The algorithm uses the following key hyperparameters:

- **Step size for policy parameters (α_θ)**: This hyperparameter determines how much the policy parameters are updated at each learning step. A larger α_θ results in faster learning but may introduce instability by making the updates too large, causing the policy to overshoot the optimal solution. On the other hand, a smaller value leads to more gradual updates, promoting stability but potentially slowing down convergence.
- **Step size for baseline parameters (α_w)**: This hyperparameter governs how quickly the baseline parameters are updated. A higher α_w enables the baseline to adapt more rapidly, which can accelerate convergence. However, it may also lead to overfitting, as the baseline might become too sensitive to short-term fluctuations in the reward signal. Smaller values help the baseline remain more robust, though it could result in slower learning.
- **Discount factor (γ)**: This parameter determines the emphasis on future rewards during the return calculation. A higher γ value places greater importance on future rewards, encouraging the agent to adopt a more long-term perspective in decision-making. This can be advantageous for problems where delayed rewards are crucial, but it may also increase computational complexity. A smaller γ focuses more on immediate rewards, making the agent more myopic and potentially faster in learning simple tasks.
- **Exploration rate (σ)**: This parameter influences the degree of exploration versus exploitation in the agent's action selection process. A higher σ increases the exploration rate, making the agent more likely to try less certain actions. As σ decreases, the agent's actions become more deterministic, focusing on exploiting known strategies that yield higher rewards. This balance is important to ensure the agent learns effectively without getting stuck in suboptimal policies.

The One-Step Actor-Critic algorithm is a simple and effective method that combines two key ideas: learning a policy (actor) and learning how good actions are (critic). The critic helps the actor by reducing noise in the learning process, making updates more stable and faster. This approach helps the algorithm learn better strategies step by step, making it a great tool for solving reinforcement learning problems.

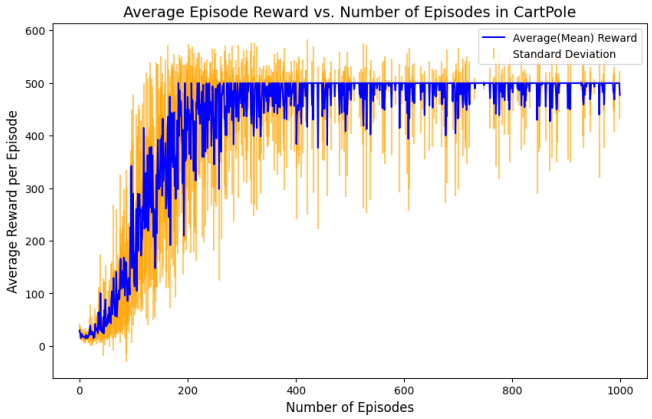
3.6 Experiments & Results : One-Step Actor-Critic

5.6.1 - Cartpole

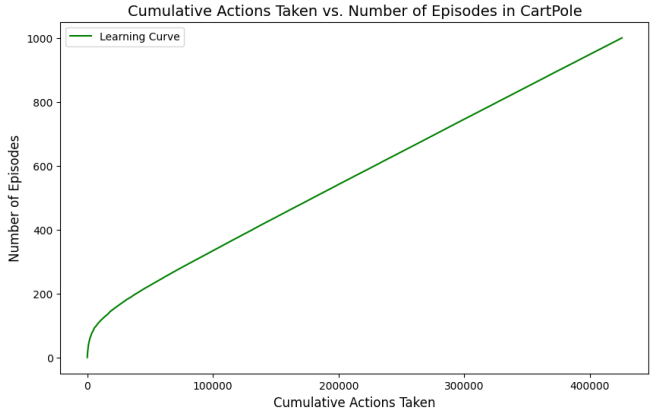
(A) Best Performing Experiment : In this experiment, we applied the One-Step Actor-Critic algorithm to the CartPole MDP. The goal was to train an agent to balance a pole on a cart by maximizing the cumulative reward over episodes. The hyperparameters used in this experiment were carefully selected based on prior exploration to achieve optimal performance. The settings for this experiment are as follows:

- Discount factor, $\gamma = 0.925$
- Step size for updating the policy, $\alpha_\theta = 0.01$

- Step size for updating the value function, $\alpha_w = 0.01$
- Epsilon (exploration rate), $\epsilon = 0.8$
- Number of training episodes, $n = 1000$



(a) Average Episode Steps vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)



(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)

Figure 19: Comparison of Average Episode Steps and Learning Curve in Cartpole.

The graph depicts the agent's performance over 1000 episodes, showing both the average reward per episode and the standard deviation. The results highlight the effectiveness of the chosen parameters in training the agent to perform consistently in the CartPole task.

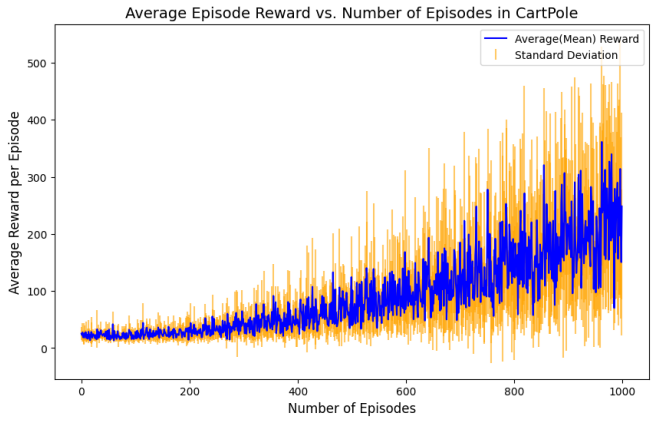
- Feature Representation: The cosine basis functions were used to approximate both the policy and value functions. This representation allowed the algorithm to capture nonlinear features of the environment efficiently, improving the learning rate and policy stability.
- Epsilon (ϵ): An epsilon value of 0.8 was chosen as the width of the basis functions. This value was found to balance the generalization and specificity of the features, leading to stable convergence.
- X-axis: Represents the number of episodes (from 0 to 1000), showing the agent's learning progression over time.
- Y-axis: Represents the average reward per episode, where higher rewards correspond to balancing the pole for a longer duration.
- Blue Line (Mean Reward): Shows the average reward achieved by the agent during training. The line's steady increase demonstrates successful learning, with the agent consistently improving its performance.
- Orange Shaded Region (Standard Deviation): Illustrates the variability in the agent's performance across episodes. Initially, the variance is large due to the agent's exploration phase, but it reduces over time, reflecting more consistent performance.

The graph illustrates the learning progression of an agent trained using the One-Step Actor-Critic algorithm in the CartPole environment. Initially, the agent experiences high variability in performance, as indicated by the steep fluctuations in rewards and large standard deviation. This reflects the agent's exploration phase, where it adapts its policy and value function through frequent updates. Such variability is typical in the early stages of reinforcement learning as the agent explores different actions and their outcomes. As training progresses, the agent's performance improves significantly, with the average reward increasing sharply and stabilizing around the maximum reward of 500 by approximately 200 episodes. This success is attributed to well-tuned hyperparameters, including the learning rates for policy and value approximation ($\alpha_\theta = \alpha_w = 0.01$) and the discount factor ($\gamma = 0.925$), which balance immediate and long-term rewards. The cosine basis function, combined with a smooth exploration rate ($\sigma = 0.8$), further enhances learning by effectively capturing the state space's structure and preventing overfitting. As training concludes, the agent's performance stabilizes, reflecting its transition from exploration to exploitation. This reduction in variability demonstrates the convergence of its learning process, with policy updates becoming more reliable and leading to

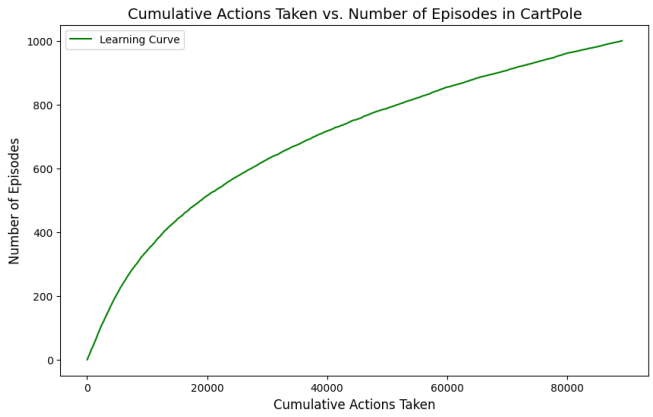
consistent performance. Overall, the graph demonstrates the effectiveness of the One-Step Actor-Critic algorithm, showcasing how a well-tuned exploration strategy and hyperparameters can optimize the agent's ability to balance the pole consistently.

Other Results

(B) We also trained the agent with a learning rate of $\alpha = 0.001$ and a discount factor of $\gamma = 0.925$ shows a slower but steady learning process compared to the agent with $\alpha = 0.01$. Initially, both agents experience some fluctuations in reward during the early episodes. However, in the case of $\alpha = 0.001$, the agent's performance improves gradually and more smoothly. In the first 400 episodes, the reward increases slowly with relatively less variability. After 400 episodes, the agent begins to show more noticeable improvements, reaching around 300 rewards by the end of 1000 episodes. On the other hand, the agent with $\alpha = 0.01$ reaches the maximum reward of 500 much earlier, around 200 episodes, and maintains this high level consistently with little fluctuation, reflecting faster convergence. This comparison indicates that a smaller learning rate results in slower learning but leads to more stable exploration, while a larger learning rate allows the agent to quickly learn the task and converge to optimal performance but with higher early fluctuations. The trade-off is evident: $\alpha = 0.001$ ensures gradual learning with fewer fluctuations, while $\alpha = 0.01$ allows for faster convergence to optimal behavior with more immediate stability.



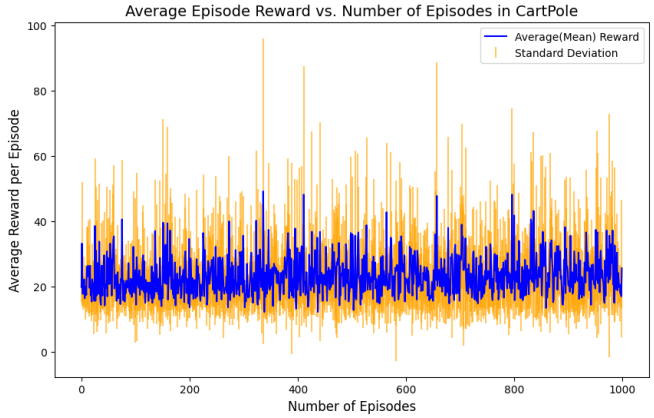
(a) Average Episode Steps vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$, $\epsilon = 0.8$)



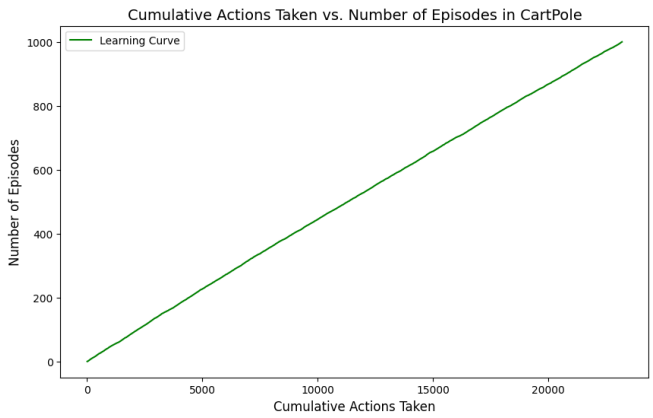
(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$, $\epsilon = 0.8$)

Figure 20: Comparison of Average Episode Steps and Learning Curve in Cartpole.

(C) We also trained the agent with a learning rate of $\alpha = 0.0001$ and a discount factor of $\gamma = 0.925$ exhibits a much slower learning curve compared to the agent trained with $\alpha = 0.01$. The graph shows that, in the early stages of training, the agent with $\alpha = 0.0001$ experiences frequent fluctuations and a lower average reward, hovering between 10 and 40 rewards per episode. As the training progresses, the rewards continue to fluctuate widely, with occasional peaks, but the agent does not seem to reach a stable or optimal performance level even after 1000 episodes. This indicates that the agent is struggling to converge to an optimal policy within the given number of episodes, likely due to the very small learning rate. In contrast, the agent with $\alpha = 0.01$ quickly reaches higher reward levels, stabilizing around 500 rewards per episode after approximately 200 episodes. This faster convergence reflects the more aggressive learning step size, allowing the agent to rapidly adapt to the environment and find an effective policy. The agent with $\alpha = 0.01$ demonstrates a clear transition from exploration to exploitation, while the agent with $\alpha = 0.0001$ shows prolonged instability and less predictable outcomes, suggesting that the learning rate needs to be higher to improve learning efficiency and stability.



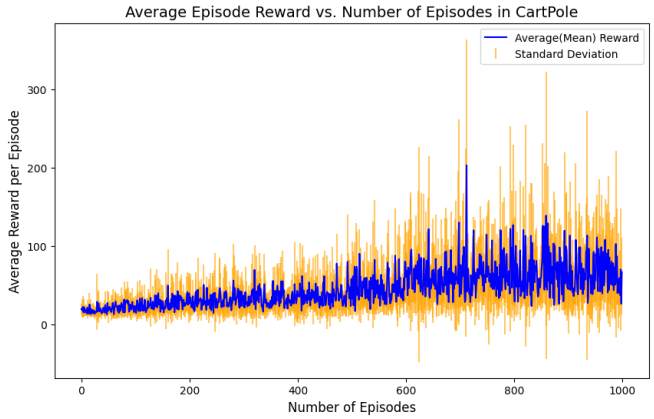
(a) Average Episode Steps vs. Number of Episodes in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$, $\epsilon = 0.8$)



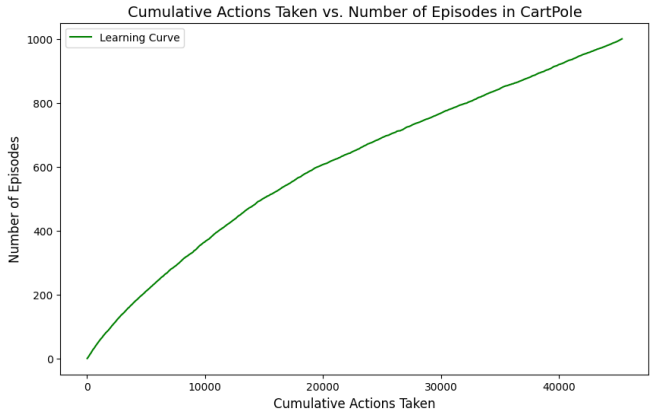
(b) Learning Curve in Cartpole ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$, $\epsilon = 0.8$)

Figure 21: Comparison of Average Episode Steps and Learning Curve in Cartpole.

(D) We also trained the agent with a learning rate of $\alpha = 0.01$ and a discount factor of $\gamma = 0.80$ shows a different learning pattern compared to the previous configurations. Initially, the agent's performance fluctuates very little, with the average reward remaining low, generally between 0 and 50 per episode. This stable but low reward range indicates that the agent is still exploring the state-action space and refining its policy. As training progresses, the agent's reward gradually increases, reflecting its improved ability to balance the pole. However, the rewards do not increase as significantly or consistently as in the previous experiments with $\gamma = 0.925$. Between episodes 600 and 800, the agent exhibits rare spikes in performance, occasionally reaching up to 200 rewards per episode, but these occurrences are infrequent and followed by a return to the 0–100 reward range. This suggests that while the agent is able to explore and find occasional effective actions, its policy has not yet fully converged to a consistently high-performing strategy. The lower γ value likely impacts the agent's ability to properly account for long-term rewards, resulting in the occasional temporary improvement but overall limited progress toward optimal performance. In contrast to the $\gamma = 0.925$ case, where the agent quickly stabilized at high rewards, this configuration leads to a more gradual and inconsistent learning curve, with lower overall performance stability across episodes.



(a) Average Episode Steps vs. Number of Episodes in Cartpole ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)



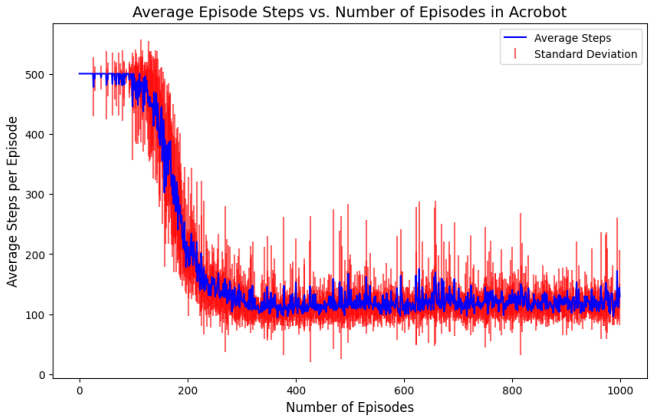
(b) Learning Curve in Cartpole ($\gamma = 0.80$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)

Figure 22: Comparison of Average Episode Steps and Learning Curve in Cartpole.

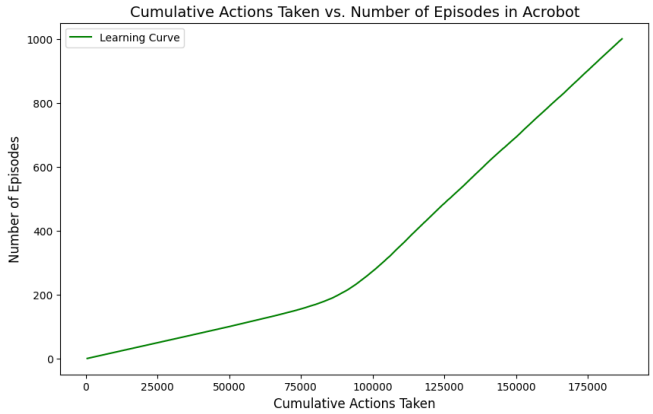
5.6.2 - Acrobot

(A) Best Performing Experiment : In this experiment, we applied the One-Step Actor-Critic algorithm to the Acrobot MDP. The goal was to train an agent to swing up and balance the Acrobot by minimizing the number of steps required to reach the desired state. The hyperparameters for this experiment were carefully chosen to ensure optimal learning and stable performance. The settings for this experiment are as follows:

- Discount factor, $\gamma = 0.925$
- Step size for updating the policy, $\alpha_\theta = 0.0001$
- Step size for updating the value function, $\alpha_w = 0.0001$
- Epsilon (exploration rate), $\epsilon = 0.8$
- Number of training episodes, $n = 1000$



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$, $\epsilon = 0.8$)



(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$, $\epsilon = 0.8$)

Figure 23: Comparison of Average Episode Steps and Learning Curve in Acrobot.

The graph depicts the agent's performance over 1000 episodes, showing both the average steps per episode and the standard deviation. The results highlight the effectiveness of the chosen parameters in training the agent to perform consistently in the Acrobot task.

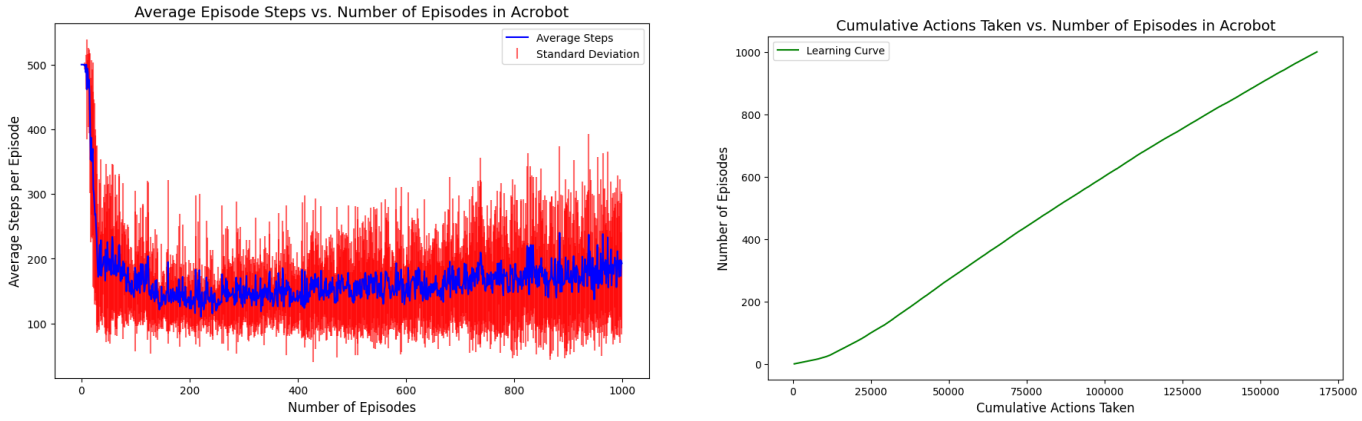
- Feature Representation: Similar to the CartPole experiment, we used cosine basis functions to approximate both the policy and value functions. This representation allowed the algorithm to capture the nonlinear features of the Acrobot environment efficiently, improving the learning rate and policy stability.
- Epsilon (ϵ): An epsilon value of 0.8 was chosen as the width of the basis functions. This value was found to balance the generalization and specificity of the features, leading to stable convergence.
- X-axis: Represents the number of episodes (from 0 to 1000), showing the agent's learning progression over time.
- Y-axis: Represents the average steps per episode, where lower steps correspond to the agent's ability to swing up and balance the Acrobot more effectively.
- **Blue Line (Mean Steps):** Represents the average number of steps per episode achieved by the agent. Initially, the agent takes a high number of steps as it explores the environment. Over time, the mean steps decrease, reflecting the agent's learning and refinement of its strategy to achieve the goal more efficiently.
- **Red Shaded Region (Standard Deviation):** Represents the variability in the agent's performance. During exploration, the standard deviation is high due to the variability in the agent's actions. As learning progresses, the standard deviation decreases, indicating more consistent performance, with occasional spikes representing brief periods of instability.

The graph illustrates the learning progression of the agent in the Acrobot environment using the One-Step Actor-Critic algorithm. Initially, the agent starts with a relatively high average number of steps, around 500, and maintains this level for the first 100 episodes. During this phase, the agent is still exploring the environment, and the standard deviation is slightly high, fluctuating by about ± 30 to ± 40 steps, which suggests some variability in the agent's actions as it explores different strategies. At around episode 100, the agent undergoes a significant drop in the number of steps, which continues until around episode 200, where the agent reaches just above the 100-step mark. This sharp decline reflects the agent's gradual learning and refinement of its policy. The large reduction in steps indicates that the agent

has started exploiting a more effective strategy, making fewer unnecessary moves to achieve the goal. After episode 200, the average number of steps stabilizes at a value just above 100. The performance remains consistent through the remaining 800 episodes, demonstrating the agent's success in mastering the task. The standard deviation decreases significantly after the initial phase of high fluctuation. While occasional spikes in performance still occur, these are less frequent and smaller in magnitude, indicating that the agent is moving toward a more stable policy with fewer large deviations. The stabilization of both the average number of steps and the standard deviation towards the later stages of training suggests that the agent has successfully learned to balance the Acrobot with minimal fluctuation in performance. The agent's learning process, characterized by the sharp decrease in steps followed by stable performance, highlights the effectiveness of the One-Step Actor-Critic algorithm in achieving consistent, optimal behavior over time.

Other Results

(B) We also trained the agent using the same learning rates for policy and value approximation ($\alpha_\theta = \alpha_w = 0.001$) and a discount factor of $\gamma = 0.925$. Initially, the agent showed a rapid decrease in the number of steps per episode, from around 500 to 200 steps within the first 10-20 episodes, indicating quick adaptation during the early exploration phase. The standard deviation remained low initially, reflecting consistent improvements in the agent's policy. However, after this early phase, the agent's performance showed more fluctuation, with the standard deviation increasing to ± 100 steps, suggesting ongoing exploration and occasional suboptimal actions as it fine-tuned its strategy. Compared to the best-performing experiment, the new experiment resulted in slightly higher steps per episode, with the agent taking a bit more time to reach optimal performance. While both experiments eventually stabilized, the new experiment showed a more gradual improvement and took longer to reach a steady state. The variance in performance was also more pronounced during the middle phase of training. This suggests that while both setups demonstrated effective learning, the new experiment exhibited more exploration in the later stages, leading to a slightly higher number of steps overall before the agent reached consistent performance.

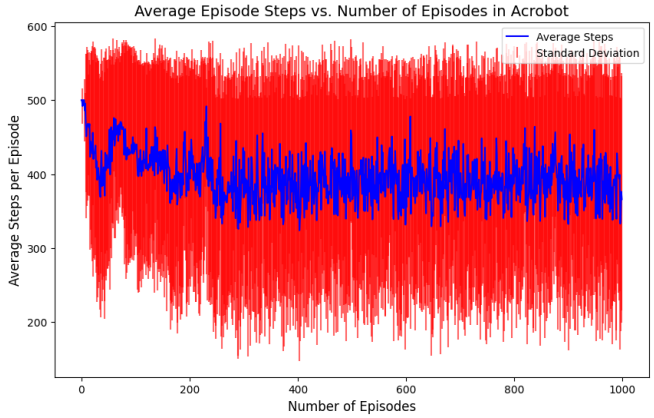


(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$, $\epsilon = 0.8$)

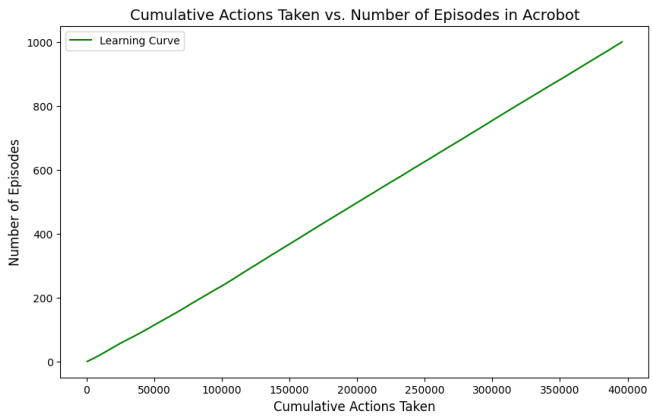
(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$, $\epsilon = 0.8$)

Figure 24: Comparison of Average Episode Steps and Learning Curve in Acrobot.

(C) We also trained the agent with the learning rates for policy and value approximation ($\alpha_\theta = \alpha_w = 0.01$) and a discount factor of $\gamma = 0.925$. This configuration exhibits a more volatile and unstable learning pattern compared to the $\alpha = 0.0001$ graph. The average number of steps starts around 500, fluctuating significantly between 350 to 450 steps in the early episodes. The standard deviation remains large, with fluctuations reaching up to 550 steps, indicating considerable variability in the agent's performance. Although the agent eventually reaches a minimum of around 200 steps, the learning process remains inconsistent due to the erratic policy updates caused by the higher learning rate. This volatility makes the $\alpha = 0.01$ configuration less reliable and stable compared to the $\alpha = 0.0001$ setup.



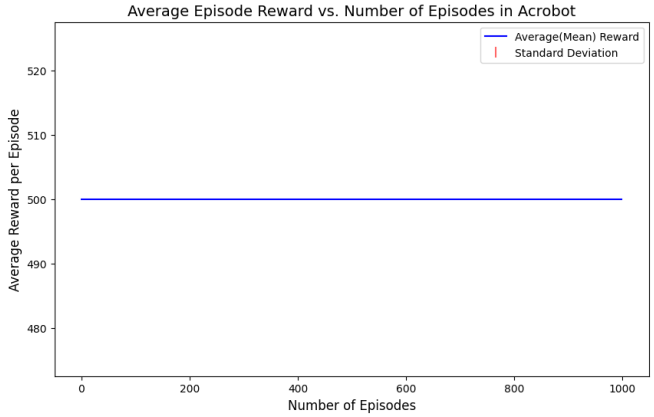
(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)



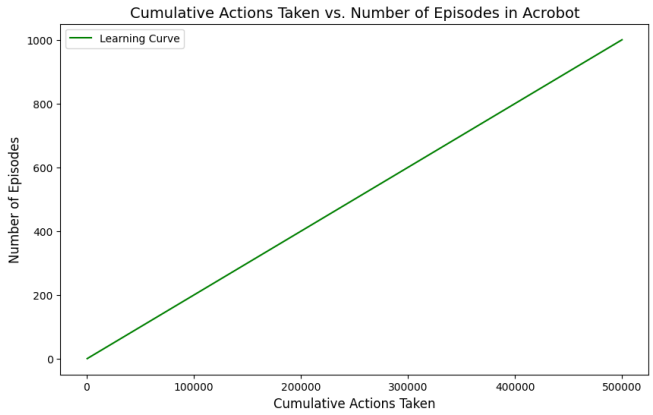
(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)

Figure 25: Comparison of Average Episode Steps and Learning Curve in Acrobot.

(D) We also trained the agent with the learning rates for policy and value approximation ($\alpha_\theta = \alpha_w = 0.3$) and a discount factor of $\gamma = 0.925$. This configuration displays an even more problematic trend where the number of steps does not decrease meaningfully throughout training, resulting in a flat, near-constant line. The agent struggles to improve its performance, showing signs of failure in training. The steps remain high and do not show the expected decrease, suggesting that the large learning rate leads to unstable policy updates and prevents effective learning. As a result, this configuration fails to show any meaningful progress in reducing the number of steps per episode.



(a) Average Episode Steps vs. Number of Episodes in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.3$, $\alpha_w = 0.3$, $\epsilon = 0.8$)



(b) Learning Curve in Acrobot ($\gamma = 0.925$, $\alpha_\theta = 0.3$, $\alpha_w = 0.3$, $\epsilon = 0.8$)

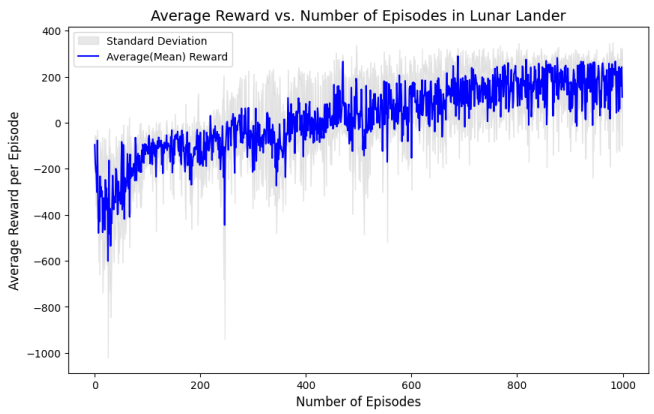
Figure 26: Comparison of Average Episode Steps and Learning Curve in Acrobot.

5.6.3 - Lunar Lander

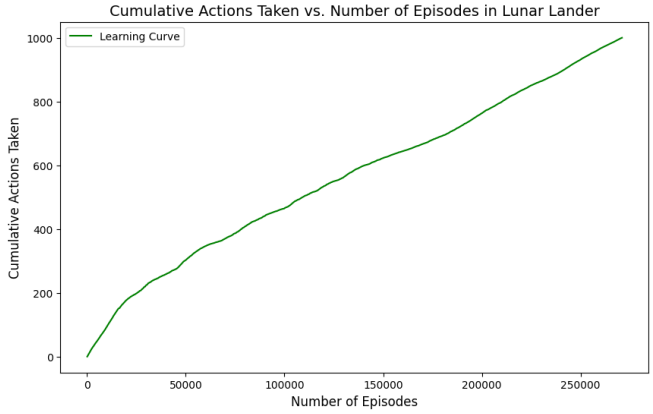
(A) Best Performing Experiment: In this experiment, we applied the One-Step Actor-Critic algorithm to the Lunar Lander environment. The goal was to train an agent to effectively land a spacecraft on a designated landing pad while maximizing cumulative rewards over 1000 episodes. The hyperparameters were fine-tuned based on prior experimentation and are as follows:

- Discount factor, $\gamma = 0.99$
- Step size for updating the policy, $\alpha_\theta = 0.01$
- Step size for updating the value function, $\alpha_w = 0.01$
- Epsilon (exploration rate), $\epsilon = 0.8$

- Number of training episodes, $n = 1000$



(a) Average Episode Reward vs. Number of Episodes in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)



(b) Learning Curve in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$, $\epsilon = 0.8$)

Figure 27: Comparison of Average Episode Reward and Learning Curve in Lunar Lander.

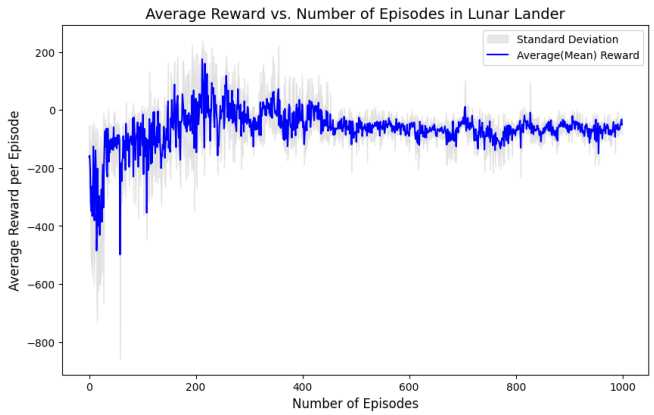
The graph illustrates the agent's performance across 1000 training episodes, with the blue line representing the average reward and the shaded region indicating the standard deviation. The results highlight the effectiveness of the chosen hyperparameters in training the agent to land successfully.

- **Feature Representation:** Similar to the Acrobot experiment, we used cosine basis functions for approximating the policy and value functions. This representation allowed the agent to effectively capture the nonlinear dynamics of the Lunar Lander environment and improve learning stability.
- **Exploration Rate (ϵ):** An exploration rate of $\epsilon = 0.8$ ensured adequate exploration of the state-action space during training. This high exploration rate facilitated learning by encouraging the agent to sample diverse trajectories.
- **X-axis:** Represents the number of episodes (from 0 to 1000), showcasing the agent's learning progression over time.
- **Y-axis:** Represents the average episode reward, where higher rewards correspond to successful landings and efficient use of fuel.
- **Blue Line (Mean Reward):** The steadily increasing mean reward signifies consistent improvement in performance. By episode 800, the agent achieves an average reward exceeding +200, demonstrating proficiency in landing the spacecraft effectively.
- **Gray Shaded Region (Standard Deviation):** The shaded area represents the standard deviation of rewards, indicating variability in the agent's performance. Initially, the variability is large due to exploration, but it narrows significantly as the agent learns a stable policy.

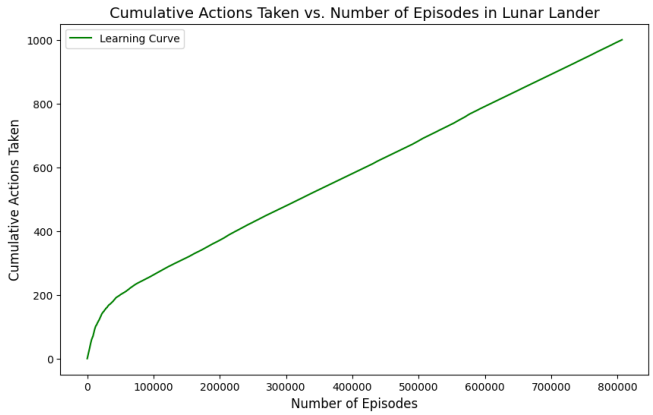
The learning curve highlights rapid progress during the initial 300 episodes, where the agent transitions from highly negative rewards (indicating frequent crashes) to rewards closer to zero. By episode 600, the rewards stabilize around +200, indicating the agent has effectively mastered the task of landing the spacecraft. Compared to early episodes, the reduced standard deviation from episode 500 onward signifies the agent's increasing reliability and consistency in its actions. This experiment demonstrates the effectiveness of the One-Step Actor-Critic algorithm in balancing policy optimization (via the actor) and value function estimation (via the critic). The selected hyperparameters ($\gamma = 0.99$, $\alpha_\theta = 0.01$, $\alpha_w = 0.01$) strike an effective balance between exploration and exploitation, enabling the agent to achieve robust performance within the given timeframe. While slight fluctuations in rewards persist due to the stochastic nature of the algorithm and environment, the overall stability and high rewards indicate the successful application of the One-Step Actor-Critic method to this problem.

Other Results

(B) In this experiment, we trained the agent with learning rates of $\alpha_\theta = \alpha_w = 0.001$ while keeping the discount factor $\gamma = 0.99$. The agent showed rapid progress, with the average reward exceeding 0 before 200 episodes, reflecting quick convergence. By episode 220, the agent reached a peak reward near 150, but following this, the performance declined and stabilized around 0 with minimal fluctuations. This suggests that while the agent converged quickly, it may have overshot optimal policies due to the higher learning rate, and eventually settled into a stable but suboptimal policy, with less variability in the later stages of training. When comparing this configuration to the best-performing experiment (REINFORCE with $\gamma = 0.99$, $\alpha_\theta = 0.01$, and $\alpha_w = 0.01$), a significant difference emerges. The best-performing agent consistently reaches rewards well above 200 by episode 900, with a stable and steady increase in performance. In contrast, the agent with $\alpha_\theta = \alpha_w = 0.001$ peaks quickly but never exceeds 150, and its performance drops after episode 220. The best-performing setup continues to improve without significant dips, while the agent with the higher learning rate struggles to maintain optimal performance. Despite faster convergence, the agent with $\alpha_\theta = \alpha_w = 0.001$ ultimately fails to match the long-term effectiveness and stability of the best-performing configuration.



(a) Average Episode Reward vs. Number of Episodes in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$, $\epsilon = 0.8$)

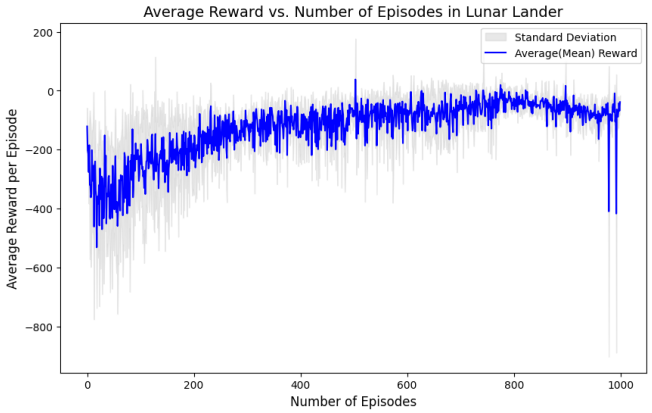


(b) Learning Curve in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.001$, $\alpha_w = 0.001$, $\epsilon = 0.8$)

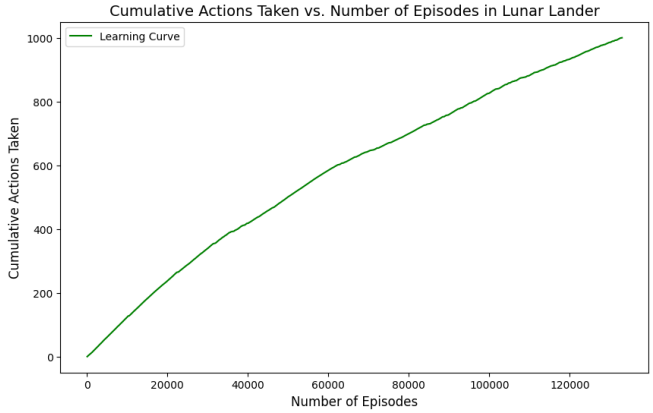
Figure 28: Comparison of Average Episode Reward and Learning Curve in Lunar Lander.

(C) When we lowered the learning rates to $\alpha_\theta = \alpha_w = 0.0001$, the agent exhibited a more gradual improvement in performance, with rewards initially near -200. The performance decreased further for the first 50 episodes but then gradually improved, reaching just below 0 around episode 400. However, the reward remained around this value for the remainder of the training, indicating that the smaller learning rates resulted in slower convergence. While the reduced fluctuations suggest more stable learning, the agent struggled to surpass the neutral reward threshold, ultimately achieving suboptimal performance. The slower updates to the policy, while promoting stability, hindered the agent's ability to effectively optimize the policy within 1000 episodes.

In comparison to the best-performing experiment, the agent with $\alpha_\theta = \alpha_w = 0.0001$ demonstrated much slower learning. The REINFORCE configuration with $\alpha_\theta = \alpha_w = 0.01$ consistently reached rewards above 200 by episode 900 and continued to improve throughout the training, whereas the agent with the smaller learning rate never exceeded 0. This comparison highlights that, while the lower learning rate results in fewer fluctuations and greater stability, it leads to much slower convergence, ultimately preventing the agent from reaching higher rewards. In contrast, the best-performing configuration not only achieved higher rewards but also demonstrated superior learning efficiency and policy optimization, consistently outperforming the agent with $\alpha_\theta = \alpha_w = 0.0001$.



(a) Average Episode Reward vs. Number of Episodes in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$, $\epsilon = 0.8$)



(b) Learning Curve in Lunar Lander ($\gamma = 0.99$, $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$, $\epsilon = 0.8$)

Figure 29: Comparison of Average Episode Reward and Learning Curve in Lunar Lander.