2024. This work is licensed under a Creative Commons “Attribution 4.0 International” license.

Exploring Advanced Reinforcement Learning Algorithms: A Comparative Study on CartPole and Acrobot Environments

Mukta Maddipatla[1] and Prudhvi Nikku[2]

[1]University of Massachusetts, Amherst
[2]University of Massachusetts, Amherst

In this project, we focus on implementing and analyzing advanced Reinforcement Learning algorithms to evaluate their performance on environments with continuous state spaces, specifically CartPole and Acrobot, using the OpenAI Gym toolkit Brockman et al. (2016). We have implemented baseline algorithms such as REINFORCE with baseline Williams (2004), the One Actor-Critic method Sutton and Barto (2018), and Semi-Gradient N-step SARSA. Furthermore, we extended our study by incorporating the Proximal Policy Optimization (PPO) algorithm Schulman et al. (2017b), a widely adopted policy gradient method. Through these implementations, we explore and compare the advantages and limitations of each approach, providing insights into their effectiveness across these environments.

1 Introduction

Reinforcement learning has gained significant attention for its ability to automate and optimize decision-making processes with minimal human supervision. Beyond traditional applications, reinforcement learning is now a critical component in advanced areas such as robotics and game AI. In this project, we explore, benchmark, and analyze the performance of four reinforcement learning algorithms: REINFORCE with baseline, One Actor-Critic method, PPO, and Semi-Gradient N-step SARSA. These algorithms are tested on environments with continuous state spaces and discrete action spaces, specifically CartPole and Acrobot, to evaluate their strengths and limitations. The implementation details, code and visualisation of the trained policies for our project are available in this repository. In the following sections, we provide a brief description of the environments and the methodologies employed.

2 Environments

2.1 CartPole

The CartPole environment, available in the OpenAI Gym, features a cart attached to a pole that the agent must balance. The objective is to keep the pole upright and within a specified range, where deviations greater than 12° from the vertical position are considered a failure, leading to the episode's termination. The environment's action space is discrete, allowing the cart to move either left or right. It employs a dense reward system, granting a reward of +1 for every time step the pole remains balanced. In our implementation, we utilize CartPole-v2, a finite-horizon environment with a maximum episode length of 500 time steps. Achieving a total reward of 500 indicates the agent has successfully mastered the task, and the episode ends.

2.2 Acrobot

The Acrobot environment, a classic control problem introduced by Sutton and Barto (2018), consists of a two-link pendulum connected in series. The system features a fixed joint at one end and an actuated joint between the two links. The objective is to apply torque to the actuated joint to swing the free end of the second link upward, reaching a target height above a defined threshold. The action space is discrete, offering three possible torque values: -1 (torque to the left), 0 (no torque), and 1 (torque to the right). Each action incurs a reward of -1 for every time step until the goal height is achieved. The environment operates as a finite-horizon Markov Decision Process (MDP), with a maximum episode length of 500 steps. As a result, the lowest possible reward for an episode is -500. The reinforcement learning algorithm's goal would be to maximize the cumulative reward by minimizing the number of steps required to achieve the target height.

3 Introduction to RL Algorithms

Reinforcement Learning (RL) algorithms can generally be categorized into model-based and model-free approaches. Model-based algorithms explicitly construct a model of the environment to plan and make decisions, with examples including Dynamic Programming and Monte Carlo Tree Search. In contrast, model-free algorithms learn directly from interaction with the environment, without relying on an explicit model. The algorithms implemented in this project are model-free, with most of them belonging to the category of policy gradient methods. These include REINFORCE with baseline, One Actor-Critic method, and Proximal Policy Optimization (PPO), which optimize policies directly by estimating gradients of expected rewards. Additionally, we have implemented Semi-Gradient N-step SARSA, a value-based method that learns action-value estimates to guide decision-making. Each of these algorithms provides unique insights into solving reinforcement learning problems effectively.

3.1 REINFORCE with Baseline

3.1.1 Introduction & Algorithm

REINFORCE with baseline is a policy gradient method that directly learns a parameterized policy to determine action selection, bypassing the need to learn action-value functions. The algorithm relies on the principle that the gradient of the expected return can be used to update the policy parameters effectively. This approach is particularly beneficial in environments with continuous state spaces, as it enables learning an optimal policy without being constrained to discrete state spaces.

This method is a Monte Carlo algorithm designed for episodic tasks. To address the high variance in the original REINFORCE algorithm and accelerate convergence, the method incorporates a baseline function for comparison. The baseline is any function independent of the specific action taken at a time step. In this project, we use the state value estimate $\hat{v}(S_t, w)$ as the baseline. The complete implementation of the algorithm is presented below:

Algorithm 1 REINFORCE with Baseline (episodic)

```
Require: A differentiable policy parameterization $\pi(a \mid s, \theta)$
Require: A differentiable state-value function parameterization $\hat{v}(s, w)$
  Parameters: Step sizes $\alpha^\theta > 0$, $\alpha^w > 0$, Discount Factor $\gamma$
  Initialize policy parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\boldsymbol{w} \in \mathbb{R}^d$ (e.g., to 0)
  for each episode do
      Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$
      for each step $t = 0, 1, \ldots, T-1$ do
          $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
          $\delta \leftarrow G - \hat{v}(S_t, \boldsymbol{w})$
          $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^w \delta \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$
          $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\theta \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta})$
      end for
  end for
```

3.1.2 Implementation Details

Our implementation of the REINFORCE with baseline algorithm utilizes separate neural networks for parameterizing the policy and value functions. This design choice offers adaptability for environments with both continuous and discrete state spaces. Specifically, the implementation leverages two distinct networks: the policy network and the value network.

The **policy network** maps states to action probabilities and consists of **two hidden layers with 128 and 64 neurons**, respectively, using ReLU activations. The output layer applies a softmax function to produce a probability distribution over the action space. The network outputs a number of logits corresponding to the size of the action space.

The **value network** estimates the state-value function $V(s)$ and also includes **two hidden layers with 128 and 64 neurons** (using ReLU activations). However, its output layer differs by producing a single value that represents the state-value estimate.

Both networks are optimized using the Adam optimizer to ensure efficient parameter updates. **Gradient clipping** with a maximum norm of 1.0 is applied to stabilize training. A **discount factor** (γ) is used to calculate the discounted returns for each episode, which form the basis for the policy and value updates.

During training, the agent interacts with the environment to generate trajectories consisting of states, actions, rewards, and state-value estimates. Actions are sampled from the policy network's output distribution using a categorical distribution. **Rewards are scaled** to improve stability during training, and advantages are calculated as the difference between the discounted returns and state-value estimates. These advantages are normalized to have zero mean and unit variance to further reduce variance in updates.

The policy loss is computed using the policy gradient formula, which incorporates the normalized advantage to guide the updates. The value loss is computed as the Mean Squared Error (MSE) between the discounted returns and the predicted state values. Both the policy and value networks are optimized separately using their respective loss functions and the Adam optimizer.

Hyperparameter tuning is performed using a grid search method to identify optimal learning rates ($\alpha_{\textbf{policy}}$ **and** $\alpha_{\textbf{value}}$), discount factor, and reward scaling factor. For environments like CartPole, convergence was achieved within 3000 episodes, whereas environments like Acrobot required more careful tuning due to the sparse reward structure. Additionally, for robustness, multiple random seeds were used to evaluate performance in Acrobot, accounting for variability in training outcomes.

The trained policies were evaluated by measuring cumulative rewards across multiple episodes. To ensure reproducibility, trained policy networks were saved for subsequent evaluations. Visualizations, including plots of mean and standard deviation of rewards, as well as cumulative steps over episodes, were generated to analyze training progress and performance trends.

This implementation showcases the effectiveness of the REINFORCE with baseline algorithm across diverse environments. By leveraging policy gradients, reducing variance through baselines, and incorporating entropy regularization in some cases (e.g., Acrobot), the approach successfully trained agents to perform well on tasks such as balancing the pole in CartPole and swinging the Acrobot to the target height.

3.1.3 CartPole

For the CartPole environment, both the policy and value networks consisted of two hidden layers with 128 and 64 neurons, respectively, using ReLU activation functions. To optimize the training process, we performed hyperparameter tuning across different learning rates for the policy and value networks. The learning rates for the policy network (α_{policy}) were tuned over $[1e^{-2}, 1e^{-3}, 0.005]$, while for the value network (α_{value}), the learning rates were tuned over $[0.005, 1e^{-3}, 0.01]$. A fixed maximum of 3000 episodes was used during the tuning process.

After tuning, the best hyperparameters were selected based on the model's ability to balance the pole effectively as training progressed. These hyperparameters were then used to train the model. Following this, the training was repeated 10 times to compute the mean and standard deviation of the reward per episode, which was visualized in subsequent plots.

The table below summarizes the best hyperparameters selected for training the CartPole environment:
The following plots (learning curves) 1 illustrate the training performance of the REINFORCE with baseline algorithm on the Cartpole environment:

Parameters	Values
Alpha Policy (α_{policy})	1e-2
Alpha Values (α_{value})	1e-3
Discount Factor (γ)	0.99
Number of Episodes	3000
Reward Scaling Factor	0.01

3.1.4 Acrobot

For the Acrobot environment, both the policy and value networks consisted of two hidden layers with 128 and 64 neurons, respectively, using ReLU activation functions. Hyperparameter tuning was performed using a grid search over various learning rates and other parameters. For the policy network (α_{policy}), the learning rates were tuned over $[1e^{-2}, 1e^{-3}, 1e^{-4}]$. For the value network (α_{value}), the learning rates were tuned over $[0.1, 1e^{-3}, 0.01]$. A discount factor (γ) of 0.99 and a reward scaling factor of 1.0 were fixed across all trials.

The training was repeated with three random seeds for each hyperparameter combination to ensure robustness, and the mean reward across the last 50 episodes was used to evaluate performance. The best hyperparameters were selected based on the model's ability to swing the Acrobot to the desired height effectively. Once the best hyperparameters were identified, the success rate was evaluated over five additional random seeds.

The table below summarizes the best hyperparameters selected for training the Acrobot environment:

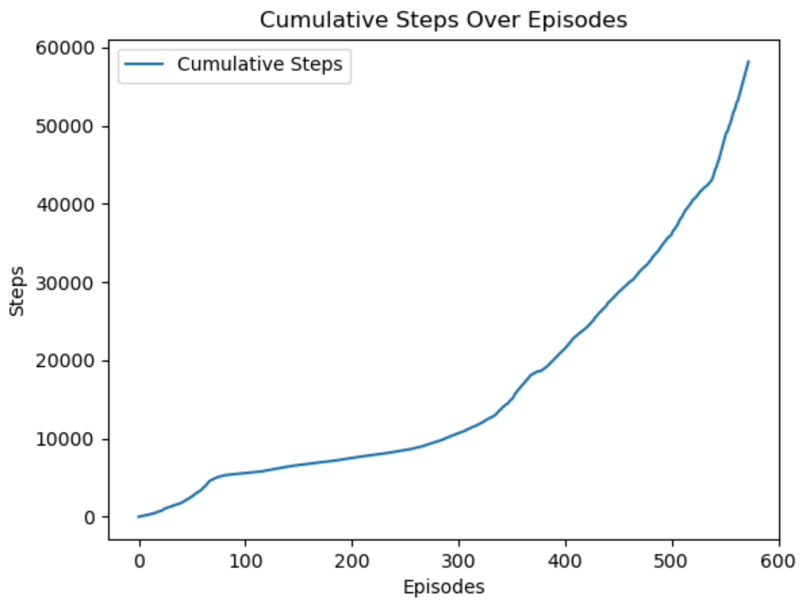
Parameters	Values
Alpha Policy (α_{policy})	1e-3
Alpha Values (α_{values})	0.01
Discount Factor (γ)	0.99
Reward Scaling Factor	1.0
Number of Episodes	3000
Random Seeds Used	3 (for tuning)

The following plots (learning curves) 2 illustrate the training performance of the REINFORCE with baseline algorithm on the Acrobot environment:

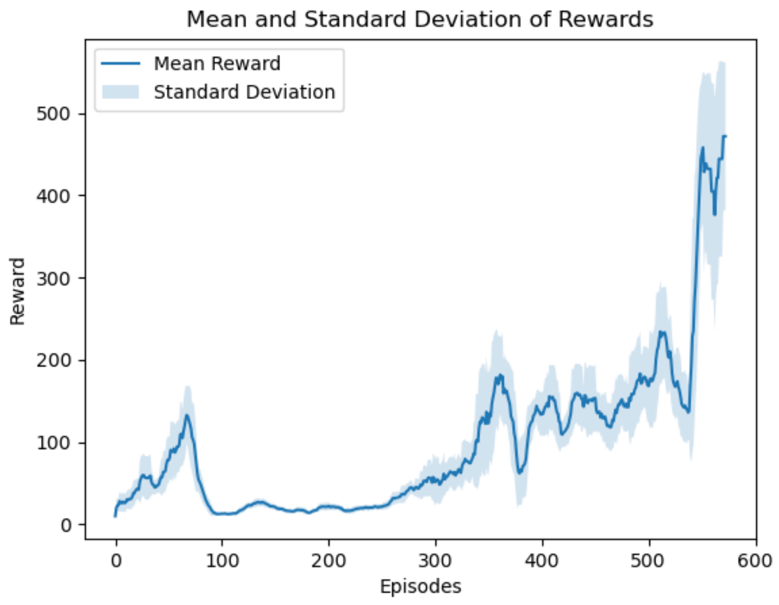
3.2 One Step Actor-Critic method

3.2.1 Introduction & Algorithm

The One-Step Actor-Critic algorithm combines policy gradient methods with value function approximation, enabling more efficient and frequent updates compared to episodic methods like REINFORCE. It employs an "actor" to select actions based on a learned policy and a "critic" to evaluate the chosen actions using a value function. By utilizing Temporal Difference (TD) error, the algorithm updates the policy and value estimates at each step, reducing variance while maintaining learning efficiency.



(a) Cumulative Steps Over Episodes.

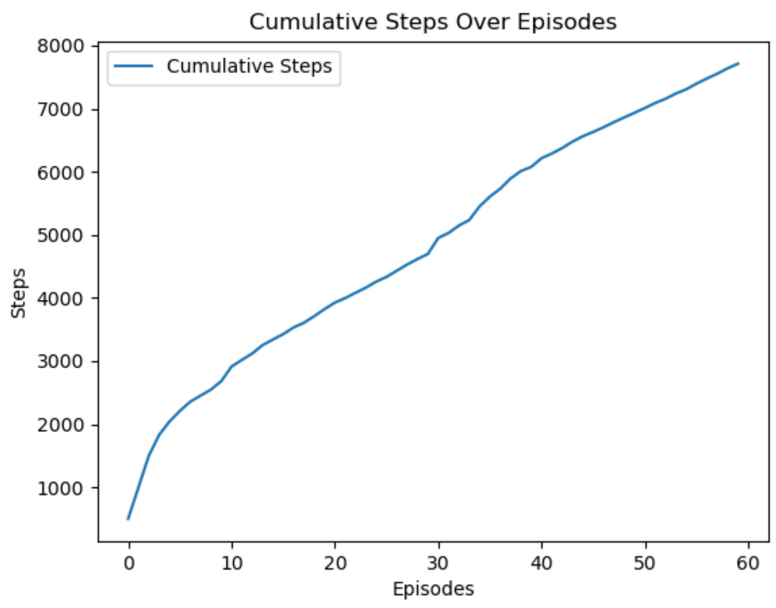


(b) Mean and Standard Deviation of Rewards.

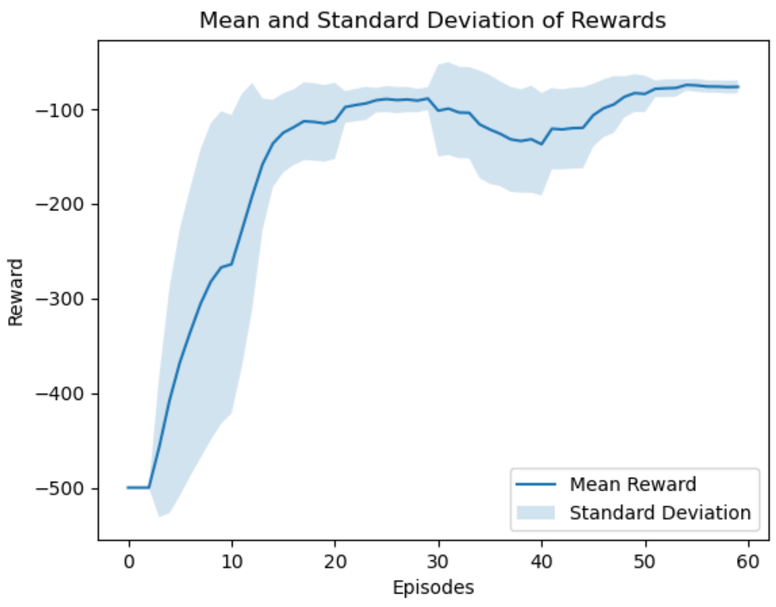
Figure 1: Training performance of the REINFORCE with baseline algorithm over Cartpole environment.

This step-by-step feedback mechanism allows the agent to adapt quickly, making the algorithm particularly effective in environments with continuous state spaces or long episodes. Its ability to balance the trade-off between bias from one-step updates and variance from sampling makes One-Step Actor-Critic a versatile and powerful reinforcement learning method.

Unlike REINFORCE, which uses the full episode return, One-Step Actor-Critic



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 2: Training performance of the REINFORCE with baseline algorithm over Acrobot environment.

replaces it with the one-step return, incorporating the reward and the estimated value of the next state. This approach reduces variance and allows updates at each step, enabling faster learning and better performance in tasks like CartPole and Acrobot, where immediate feedback significantly improves training stability.

Below is the pseudocode for One Step Actor Critic Algorithm :

Algorithm 2 One-step Actor-Critic (episodic)

```
Require: A differentiable policy parameterization $\pi(a \mid s, \theta)$
Require: A differentiable state-value parameterization $\hat{v}(s, w)$
  Parameters: Step sizes $\alpha^\theta > 0, \alpha^w > 0$, Discount Factor $\gamma$, Network Structure
  Initialize policy parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\boldsymbol{w} \in \mathbb{R}^d$
  for each episode do
      Initialize $S$ (first state of episode)
      $I \leftarrow 1$
      while $S$ is not terminal do
          $A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$
          Take action $A$, observe $S'$, $R$
          $\delta \leftarrow R + \gamma\hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$
          $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^w I\delta\nabla_{\boldsymbol{w}}\hat{v}(S, \boldsymbol{w})$
          $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\theta I\delta\nabla_{\boldsymbol{\theta}} \ln \pi(A \mid S, \boldsymbol{\theta})$
          $I \leftarrow \gamma I$
          $S \leftarrow S'$
      end while
  end for
```

3.2.2 Implementation Details

Our implementation of the One-Step Actor-Critic algorithm employs separate neural networks for parameterizing the policy (actor) and the value function (critic). This design facilitates efficient updates and is adaptable to environments with continuous state spaces and discrete action spaces. Specifically, the implementation uses two distinct networks: the policy network and the value network.

The policy network maps states to action probabilities and consists of a fully connected architecture with a single hidden layer. The number of neurons in the hidden layer is configurable (e.g., 128 in some experiments) and uses ReLU activations for non-linearity. The output layer applies a softmax function to produce a probability distribution over the action space, enabling stochastic action selection.

The value network, tasked with estimating the state-value function $V(s)$, has a similar structure with one hidden layer and ReLU activations. Unlike the policy network, the value network's output layer produces a single value representing the estimated return for a given state. Both networks are optimized independently using the Adam optimizer, with distinct learning rates (α_{policy} and α_{value}) for the actor and critic to ensure stable updates.

During training, the agent interacts with the environment step-by-step, generating trajectories of states, actions, rewards, and next states. Actions are sampled from the policy network's output distribution, allowing for exploration. The Temporal Difference (TD) error is computed as

$$\delta = r + \gamma V(s') - V(s),$$

where r is the reward, γ is the discount factor, s' is the next state, and $V(s)$ is the current state-value estimate. The TD error drives updates for both networks, enabling the critic to minimize this error and the actor to adjust its policy in the direction of higher rewards.

The policy loss is calculated using the negative log-likelihood of the chosen action weighted by the TD error, encouraging actions that lead to higher returns. The value loss is computed as the Mean Squared Error (MSE) between the estimated and actual returns, enabling more accurate value predictions. The networks are updated step-by-step, reducing variance and allowing faster convergence compared to episodic methods.

Hyperparameter tuning is conducted to identify optimal learning rates, hidden layer sizes, and discount factors for different environments. In experiments with environments like CartPole and Acrobot, the algorithm demonstrated robust performance.

Evaluation metrics include cumulative rewards and average steps across multiple episodes. Training progress is visualized through plots of mean and standard deviation of rewards, along with cumulative actions taken per episode. These insights highlight the One-Step Actor-Critic algorithm's efficiency and adaptability in solving reinforcement learning tasks.

3.2.3 CartPole

The Actor-Critic algorithm was implemented on the CartPole environment, and the best hyperparameters were determined through an extensive exploration of the parameter space. The chosen configuration of **128 neurons** in the hidden layers, a discount factor (γ) of **0.99**, a policy learning rate (α_{policy}) of **0.0001**, and a value learning rate (α_{value}) of **0.001**, with **1000 episodes** and **10 runs**, resulted in the agent achieving optimal performance. Below is a detailed analysis:

Network Architecture:- Both the policy and value networks were configured with a single hidden layer comprising **128 neurons**. This larger network size provided enhanced capacity to model the environment's dynamics and optimize the policy effectively. While smaller networks (e.g., **64, 32 neurons**) provided computational efficiency, the use of 128 neurons yielded better performance by achieving faster convergence and higher rewards.

Discount Factor (γ):- A discount factor of **0.99** enabled the agent to prioritize long-term rewards, which is essential in the CartPole environment where balancing the pole requires strategic planning. Lower discount factors (e.g., $\gamma = 0.1$, $\gamma = 0.5$) led to suboptimal policies, as the agent focused excessively on immediate rewards. The choice of $\gamma = 0.99$ allowed the agent to learn a stable and effective policy over the course of training.

Policy Learning Rate (α_{policy}):- A policy learning rate of **0.0001** provided a balance between stability and convergence speed. Higher learning rates (e.g., $\alpha_{\text{policy}} = 0.01$, $\alpha_{\text{policy}} = 0.1$) caused fluctuations in performance due to aggressive updates, while smaller rates resulted in excessively slow learning progress. The chosen rate of 0.0001 ensured steady and stable optimization.

Value Learning Rate (α_{value}):- The value learning rate of **0.001** allowed for accurate updates to the value function, enabling the critic to provide reliable feedback to the actor. While a larger learning rate ($\alpha_{\text{value}} = 0.01$,$\alpha_{\text{value}} = 0.1$) caused instability, the chosen value of 0.001 balanced learning speed with stability, ensuring efficient convergence.

Performance Evaluation:- The agent's performance was evaluated based on total actions and average rewards over 1000 episodes and 10 runs. The results demonstrate:

- **Total Actions**: The total actions per episode grew consistently, reaching over 175,000 by the 1000th episode. This indicates that the agent successfully learned to balance the pole for extended periods, maximizing the episode duration.
- **Average Rewards**: The mean reward improved significantly, reaching values over 400 by the end of training. The standard deviation narrowed as training progressed, highlighting stable and consistent learning.

Summary of Best Parameters

Parameter	**Optimal Value**
Hidden Layer Size	128 neurons
Discount Factor (γ)	0.99
Policy Learning Rate (α_{policy})	0.0001
Value Learning Rate (α_{value})	0.001
Number of Episodes	1000
Runs	10

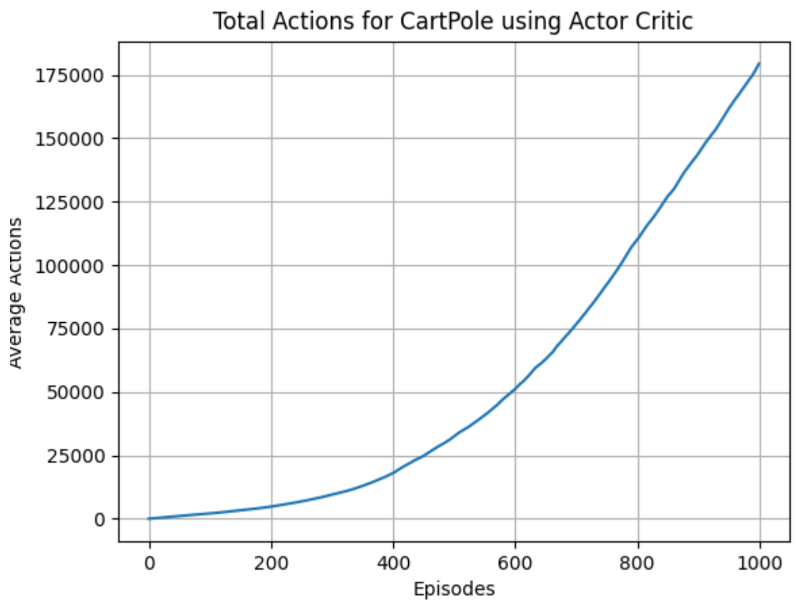
Figure 3 illustrate the training performance of the One Step Actor Critic algorithm on the Cartpole environment:

3.2.4 Acrobot

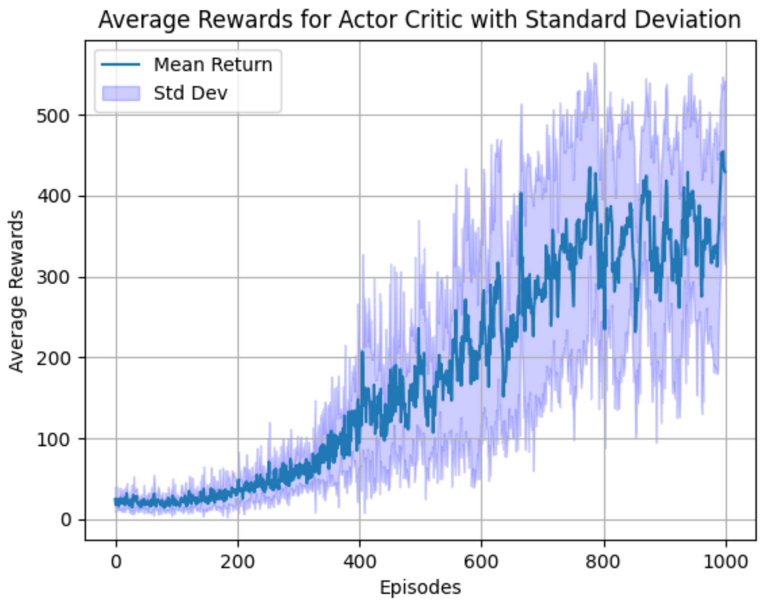
The Actor-Critic algorithm was implemented on the Acrobot environment to analyze its performance under varying hyperparameters. After extensive experimentation, the optimal configuration was determined to be **128 neurons** in the hidden layers, a discount factor (γ) of **0.99**, a policy learning rate (α_{policy}) of **0.0001**, and a value learning rate (α_{value}) of **0.001**, with **1000 episodes** and **10 runs**. Below is the detailed analysis:

Network Architecture:- The policy and value networks were configured with a single hidden layer of **128 neurons**. This configuration provided the required capacity to model the complex state-action relationships in the Acrobot environment. Networks with smaller hidden layers (e.g., **32** or **64 neurons**) displayed slower learning and suboptimal policies. Increasing the network size to 128 neurons yielded significant improvements in convergence speed and stability.

Discount Factor (γ):- A discount factor of **0.99** was optimal, allowing the agent to prioritize long-term rewards critical for reaching the target state in the Acrobot environment. Lower values (e.g., $\gamma = 0.1$) led to short-sighted policies that failed to balance long-term planning with immediate rewards. The choice of $\gamma = 0.99$ demonstrated the importance of long-term reward optimization in environments with



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 3: Training performance of the One Step Actor Critic algorithm over Cartpole environment.

sparse rewards.

Policy Learning Rate (α_{policy}):- The policy learning rate of **0.0001** ensured stable and steady updates to the policy parameters. Larger learning rates (e.g., $\alpha_{\text{policy}} = 0.001$ or $\alpha_{\text{policy}} = 0.01$) caused oscillations and instability during training, while smaller rates slowed down learning progress. The selected value of 0.0001 achieved a balance between stability and learning efficiency.

Value Learning Rate (α_{value}):- A value learning rate of **0.001** allowed the critic network to provide accurate feedback to the actor. Smaller learning rates (e.g., $\alpha_{\text{value}} = 0.0001$) delayed updates, reducing the critic's effectiveness, while higher rates (e.g., $\alpha_{\text{value}} = 0.01$) caused fluctuations in value estimates. The chosen value enabled faster and stable learning of the value function.

Performance Evaluation:- The agent's performance was evaluated over **1000 episodes** and **10 runs**. The following observations summarize the results:

- **Average Rewards**: The rewards improved steadily across episodes, with the agent achieving near-optimal returns by the end of training. The standard deviation narrowed over time, reflecting increased stability in the learned policy.
- **Learning Stability**: The agent consistently demonstrated improved performance with the selected parameters, indicating the robustness of the chosen configuration.

Summary of Best Parameters

Parameter	**Optimal Value**
Hidden Layer Size	128 neurons
Discount Factor (γ)	0.99
Policy Learning Rate (α_{policy})	0.0001
Value Learning Rate (α_{value})	0.001
Number of Episodes	1000
Runs	10

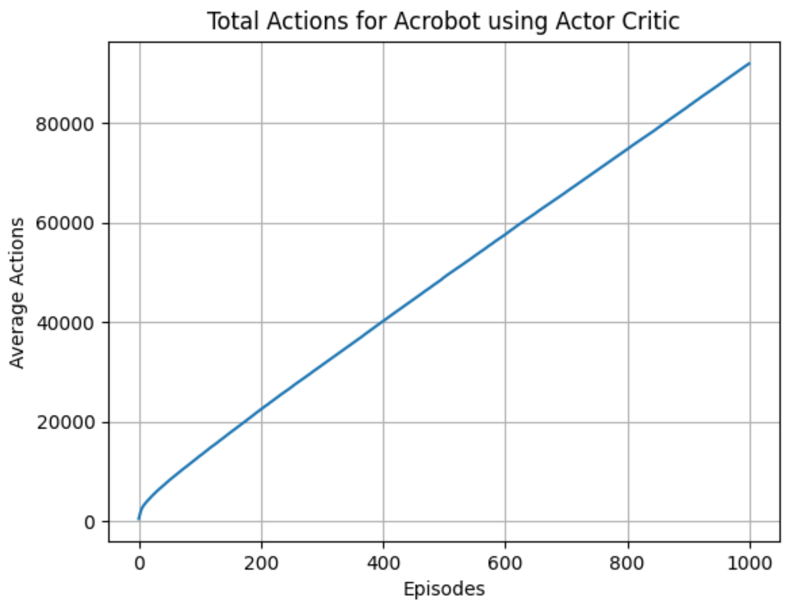
Figure 4 illustrate the training performance of the One Step Actor Critic algorithm on the Acrobot environment:

3.3 [Extra Credit] Proximal Policy Optimization (PPO)

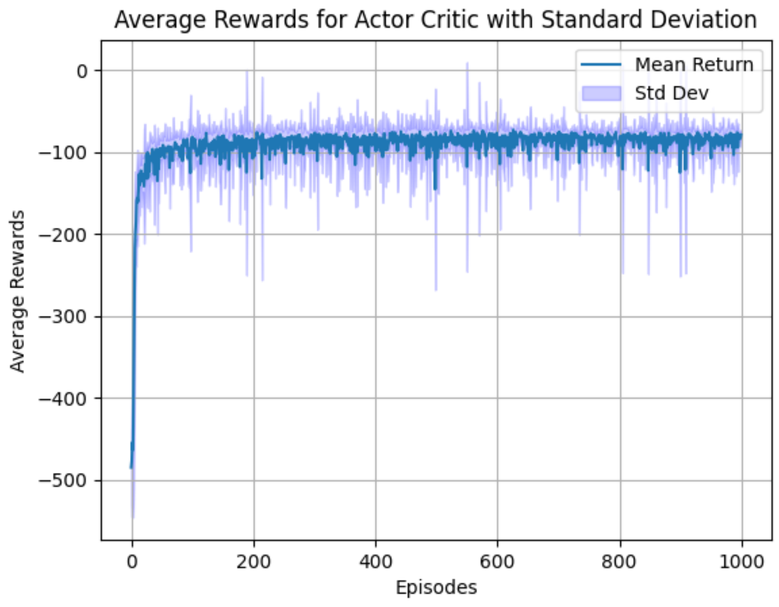
3.3.1 Introduction & Algorithm

Proximal Policy Optimization (PPO) is a popular algorithm due to its practical success in aligning reinforcement learning models with real-world requirements (e.g., aligning language models with human preferences). Policy Gradient methods, while effective, often suffer from instability, leading to updates that are either excessively large or too small, depending on the state. Trust Region Policy Optimization (TRPO) (Schulman et al., 2017a) addressed this issue by enforcing a strict constraint on the KL-divergence between successive policies, ensuring stable updates. However, TRPO's implementation can be computationally expensive and complex.

PPO simplifies this process by introducing a clipped surrogate objective function, making it more efficient and easier to implement while still maintaining stable updates. The method optimizes a surrogate objective $L_{\text{CPI}}(\theta) = \mathbb{E}_t[r_t(\theta)\hat{A}_t]$, where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, and $\hat{A}_t$ is the advantage estimate. Unlike TRPO, which enforces a strict KL-divergence constraint, PPO uses a clipped objective $L_{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$. This clipping prevents overly large updates that could destabilize training and ensures steady learning progress.



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 4: Training performance of the One Step Actor Critic algorithm over Acrobot environment.

In addition to the clipped objective, PPO incorporates the value function error and an optional entropy bonus to encourage exploration. The complete objective becomes $L_{\text{CLIP+VF+S}}(\theta) = \mathbb{E}_t[L_{\text{CLIP}}(\theta)] - c_1 \cdot L_{\text{VF}}(\theta) + c_2 \cdot S[\pi_\theta](s_t)$, where L_{VF} is the squared-error loss for the value function, $S[\pi_\theta]$ denotes the entropy bonus, and c_1 and c_2 are coefficients to balance the components.

Training PPO involves collecting trajectories, calculating the advantage estimates, and performing batch gradient descent over multiple epochs. While the PPO paper introduces Generalized Advantage Estimation (GAE) for more stable and accurate advantage calculations, our implementation simplifies this by using the standard discounted return subtracted by the current state value estimate as the advantage estimator. This approach balances computational efficiency with effective learning.

The algorithm for our implementation is as follows:

Algorithm 3 Proximal Policy Optimization (PPO)

```
Require: Policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$ and value function parameterization $V_\phi(s)$
  Parameters: Step sizes $\alpha^\theta > 0$, $\alpha^\phi > 0$, number of epochs $K$, mini-batch size $M$, discount factor $\gamma$, clipping parameter $\epsilon$, coefficients $c_1, c_2$
  Initialize policy parameters $\boldsymbol{\theta}$ and value function parameters $\phi$
  for each iteration do
      Collect a set of trajectories $D = \{S_i, A_i, R_i, \pi_\theta(A_i \mid S_i)\}$ using $\pi_\theta$
      Compute discounted returns $G_i = \sum_{t=i}^{T} \gamma^{t-i} R_t$
      Compute advantages $A(S_i, A_i) = G_i - V_\phi(S_i)$
      Create dataset $C = \{S_i, A_i, G_i, A(S_i, A_i), \pi_\theta(A_i \mid S_i)\}$
      for $k = 1$ to $K$ (number of epochs) do
          for each mini-batch $B$ of size $M$ in $C$ do
              Compute ratio $r_t(\boldsymbol{\theta}) = \frac{\pi_\theta(A|S)}{\pi_{\theta_{\text{old}}}(A|S)}$
              Compute clipped surrogate objective:
                  $L_{\text{CLIP}}(\boldsymbol{\theta}) = \mathbb{E}\left[\min\left(r_t(\boldsymbol{\theta})A(S,A), \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)A(S,A)\right)\right]$
              Update policy:
                  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\theta \nabla_{\boldsymbol{\theta}} \left(L_{\text{CLIP}}(\boldsymbol{\theta}) - c_2 \cdot S[\pi_\theta](S)\right)$
              Update value function:
                  $\phi \leftarrow \phi + \alpha^\phi \nabla_\phi \left(-c_1 \cdot (V_\phi(S) - G)^2\right)$
          end for
      end for
  end for
```

3.3.2 Implementation Details

For both the CartPole and Acrobot environments, we used distinct policy and value networks, each consisting of two hidden layers with 128 and 64 neurons, respectively. Both networks utilized ReLU activation functions, with the policy network outputting a softmax probability distribution over the action space, while the value network produced a single scalar output to estimate the state value $V(s)$. Adam optimizers were employed for training both networks, with separate learning rates tuned through grid search.

To stabilize training, we normalized the advantage estimates across all environments, ensuring numerical stability and improving convergence. The clipping parameter ϵ

was set to 0.2, consistent with the PPO paper, to prevent large policy updates. We conducted training using a mini-batch size of $M = 32$ for CartPole and $M = 256$ for Acrobot, and each batch was optimized for 10 epochs. The discount factor γ was set to 0.99, as is standard for episodic reinforcement learning tasks.

3.3.3 CartPole

The PPO algorithm demonstrated exceptional performance on the CartPole-v1 environment, learning the optimal behavior within approximately 250 episodes. Compared to other reinforcement learning algorithms, PPO exhibited significantly lower variance, resulting in stable and consistent performance across multiple runs. The training curves, as shown in Figure 5, highlight the rapid convergence of PPO in this environment.

The hyperparameters used to train the model effectively are detailed below:

Parameters	**Values**
Alpha Policy (α_θ)	1×10^{-3}
Alpha Value (α_ϕ)	5×10^{-3}
Discount Factor (γ)	0.99
Mini-batch Size (M)	32
Number of Epochs (K)	10
Clipping Parameter (ϵ)	0.2

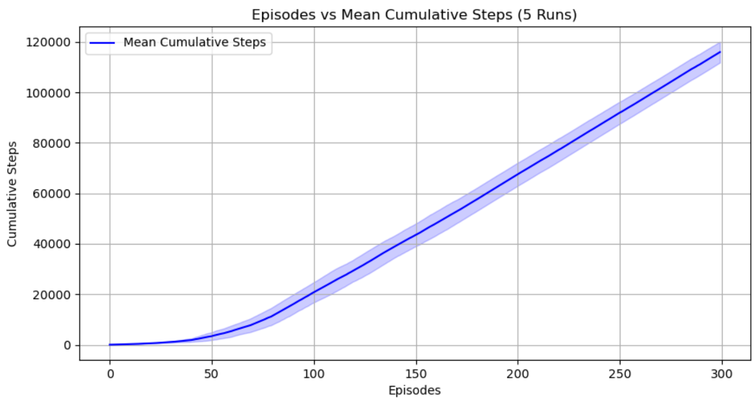
Table 1: Hyperparameters used for PPO in the CartPole-v1 environment.

By leveraging these hyperparameters, PPO efficiently learned to maximize rewards, achieving the solved score of 200 within the set horizon of 501 timesteps. The training process also demonstrated remarkable robustness, as the agent consistently attained optimal performance across multiple runs.

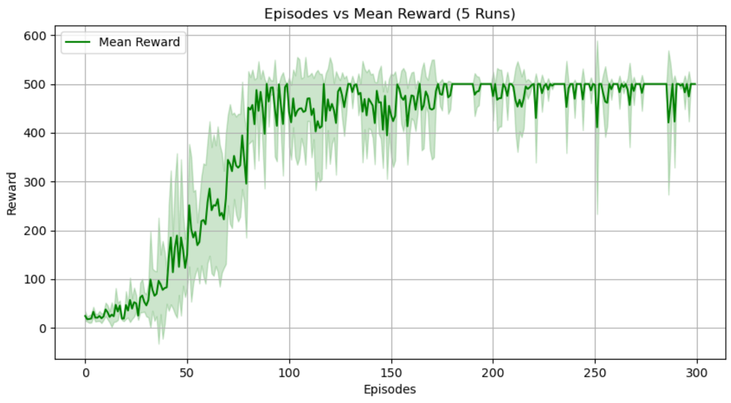
3.3.4 Acrobot

The PPO algorithm demonstrated robust performance on the Acrobot-v1 environment, which is characterized by its sparse rewards and increased difficulty. The agent successfully learned to swing the lower link of the Acrobot system to the desired height within 500 episodes. As observed in Figure 6, the cumulative steps increased steadily with episodes, while the reward stabilized around -100, reflecting the completion of the task.

The following hyperparameters were identified as optimal for training the PPO algorithm on the Acrobot environment:



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 5: Training performance of the PPO algorithm over Cartpole environment.

Parameter	**Value**
Alpha Policy (α_θ)	1×10^{-3}
Alpha Value (α_ϕ)	5×10^{-3}
Discount Factor (γ)	0.99
Mini-batch Size (M)	256
Number of Epochs (K)	10
Clipping Parameter (ϵ)	0.2

Table 2: Hyperparameters used for PPO in the Acrobot-v1 environment.

Despite the sparsity of rewards, PPO effectively stabilized training through the use of clipped surrogate objectives and advantage normalization.The agent achieved an average reward of -100, indicating task success, with low variability in performance across five independent runs.The graph of cumulative steps over episodes showed a consistent increase, indicating steady learning progress.The learning curves for PPO over Acrobot: 6

The combination of well-tuned hyperparameters and PPO's robust optimization techniques allowed the agent to overcome the challenges of sparse rewards in the

Acrobot-v1 environment efficiently. These results further affirm the capability of PPO to handle complex environments with high-dimensional action spaces.

Advantage estimates were computed as the difference between discounted returns and state value predictions, ensuring the policy updates were guided by accurate estimates of advantage. For the Acrobot environment, minor reward shaping was introduced to encourage angular velocity toward the goal and penalize deviations. Across multiple runs with different random seeds, PPO demonstrated low variance in CartPole but exhibited moderate variability in Acrobot due to the sparse reward structure. The results highlight the need for careful hyperparameter tuning, particularly for ϵ and learning rates, to balance exploration and exploitation effectively. Figures 5 and 6 illustrate PPO's rapid convergence and stability in both environments, affirming its robustness in dense and sparse reward settings.

3.4 [Extra Credit] Semi-Gradient N-step SARSA

3.4.1 Introduction & Algorithm

The **Semi-Gradient N-Step SARSA** algorithm that extends the classic SARSA method by incorporating multi-step returns and approximating the action-value function using a parameterized model. By leveraging the semi-gradient method, this algorithm optimizes the action-value function with respect to weights of a neural network, enabling it to scale efficiently to environments with large or continuous state-action spaces. The use of N-step returns allows the agent to balance the trade-off between short-term and long-term rewards, providing better learning stability compared to one-step methods.

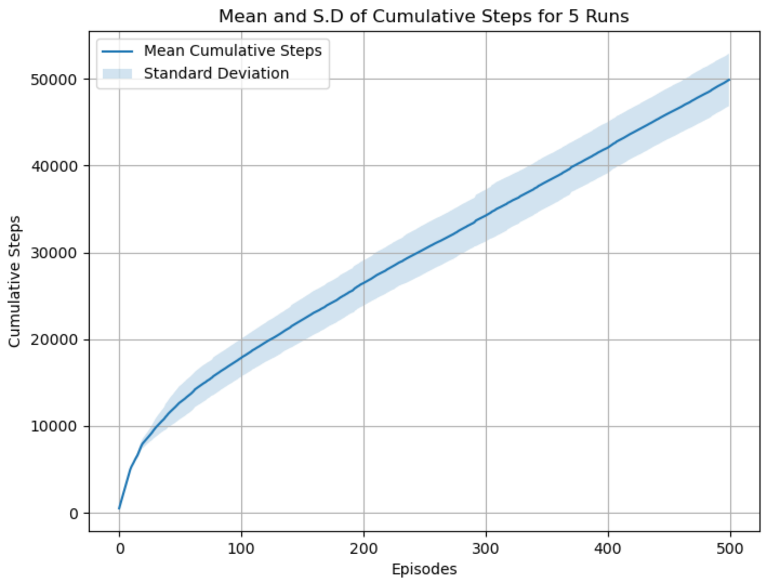
Unlike standard SARSA, which relies on single-step updates, the Semi-Gradient N-Step SARSA uses a temporal difference (TD) learning approach to compute the discounted cumulative reward over multiple future steps. This reduces bias and allows the algorithm to converge more effectively. Furthermore, the use of an ϵ-greedy policy ensures sufficient exploration, preventing the agent from converging prematurely to suboptimal policies. By tuning hyperparameters such as the learning rate, discount factor, and the number of steps n, this algorithm achieves robust performance across different reinforcement learning tasks.

Below is the pseudocode for the Semi-Gradient N-Step SARSA algorithm:

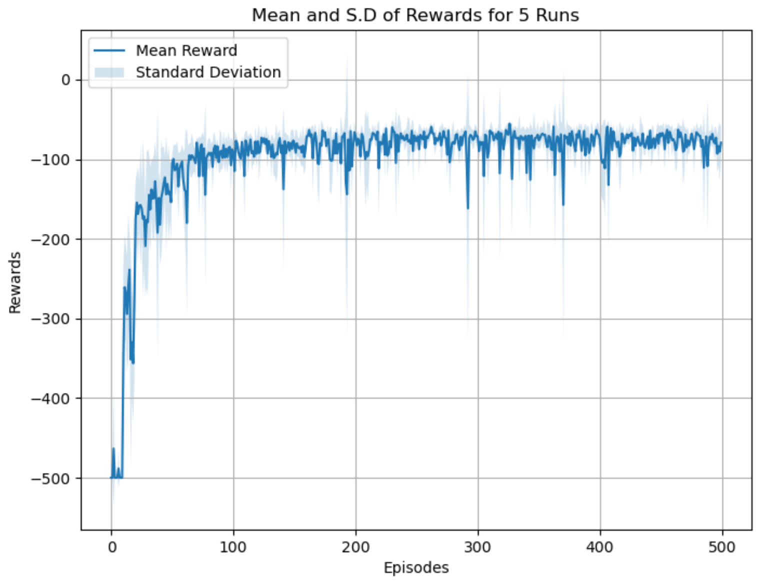
3.4.2 Implementation Details

Our implementation of the Semi-Gradient n-Step SARSA algorithm employs a neural network to approximate the action-value function (q). This design is highly flexible and adaptable to environments with both continuous and discrete state spaces. The algorithm integrates a parametric function approximator for $q(s, a)$ to estimate the expected return for state-action pairs efficiently.

The action-value network consists of a fully connected architecture with a single hidden layer. The number of neurons in the hidden layer is configurable (e.g., 128 in our experiments) and uses ReLU activations for non-linearity. The input layer concatenates the state and action vectors, allowing the network to learn representations for state-action dependencies. The output layer produces a single



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 6: Training performance of the PPO algorithm over Acrobot environment.

scalar value representing the estimated action-value function, enabling efficient computation of $q(s, a)$.

During training, the algorithm iteratively interacts with the environment to generate trajectories of states, actions, rewards, and next states. At each time step, the agent selects actions based on an ϵ-greedy policy derived from the current estimates of $q(s, a)$. The algorithm maintains a fixed-length buffer to store state-action-reward

Algorithm 4 Episodic semi-gradient n-step SARSA for estimating $\hat{q} \approx q_*$ or q_π

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, small $\epsilon > 0$, a positive integer n
Initialize: value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = 0$)
All store and access operations (S_t, A_t, and R_t) can take their index mod $n + 1$

```
1:  loopfor each episode:
2:      Initialize and store $S_0 \neq$ terminal
3:      Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or $\epsilon$-greedy w.r.t. $\hat{q}(S_0, \cdot, \mathbf{w})$
4:      $T \leftarrow \infty$
5:      for $t = 0, 1, 2, \ldots$ do
6:          if $t < T$ then
7:              Take action $A_t$
8:              Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
9:              if $S_{t+1}$ is terminal, then: then
10:                 $T \leftarrow t + 1$
11:             else
12:                 Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or $\epsilon$-greedy w.r.t. $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
13:             end if
14:         end if
15:         $\tau \leftarrow t - n + 1$                ▷ $\tau$ is the time whose estimate is being updated
16:         if $\tau \geq 0$ then
17:             $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
18:             if $\tau + n < T$, then: then
19:                 $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$
20:             end if
21:             $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[ G - \hat{q}(S_\tau, A_\tau, \mathbf{w}) \right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$
22:         end if
23:         if $\tau = T - 1$ then
24:             break
25:         end if
26:     end for
27: end loop
```

sequences, which are used to compute the n-step return (G).

The n-step return is calculated as:

$$G = \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i + \gamma^n q(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) \quad (\text{if } \tau + n < T),$$

where γ is the discount factor, T is the terminal time step, and $\mathbf{w}$ represents the neural network's parameters. For terminal states, the target G excludes the bootstrapped term.

The loss function minimizes the temporal difference (TD) error:

$$\delta = G - q(S_\tau, A_\tau, \mathbf{w}).$$

The neural network's weights are updated using gradient descent to minimize the mean squared TD error. The Adam optimizer is employed with a learning rate α to ensure stable updates during training.

Hyperparameter tuning is conducted to identify the optimal learning rates, hidden layer sizes, discount factors, and the step size (n) for different environments. In experiments with environments like CartPole and Acrobot, the Semi-Gradient n-Step SARSA algorithm demonstrated robust performance and stable convergence.

Evaluation metrics include cumulative rewards and average actions per episode. Training progress is visualized through plots of mean and standard deviation of rewards, along with cumulative actions per episode. These insights highlight the algorithm's efficiency and adaptability in solving reinforcement learning tasks.

3.4.3 CartPole

The Semi-Gradient N-Step SARSA algorithm was implemented on the CartPole environment, and the best hyperparameters were determined through an extensive exploration of the parameter space. The chosen configuration of 128 neurons in the hidden layer, a discount factor (γ) of 0.99, a learning rate (α) of 0.001, an exploration rate (ϵ) of 0.1, and n-step value of 5, with 1000 episodes and 5 runs, resulted in the agent achieving optimal performance. Below is a detailed analysis:

Network Architecture

The network used for the action-value function approximation consisted of a single hidden layer with 128 neurons, using ReLU activation functions. This configuration provided sufficient capacity to model the environment's dynamics effectively while maintaining computational efficiency. The Adam optimizer was employed for training with the selected learning rates. We experimented with various neurons (e.g., 64, 128,256 neurons). Smaller networks couldn't give good results as it couldnt represent the Q network optimally and larger network took more time to converge.

Discount Factor (γ)

A discount factor of 0.99 enabled the agent to prioritize long-term rewards, which is critical in the CartPole environment where balancing the pole requires strategic planning. Lower discount factors (e.g., $\gamma = 0.4$, $\gamma = 0.8$) led to suboptimal policies as the agent focused excessively on immediate rewards. The choice of $\gamma = 0.99$ allowed the agent to learn a stable and effective policy over the course of training.

Learning Rate (α)

The learning rate of 0.001 allowed for stable updates to the action-value function. Higher learning rates (e.g., $\alpha = 0.1$) caused instability due to aggressive updates, while smaller rates (e.g., $\alpha = 0.0001$) resulted in excessively slow learning progress. The chosen value of $\alpha = 0.001$ balanced learning speed with convergence stability.

Exploration Rate (ϵ)

An exploration rate of 0.1 provided a balance between exploration and exploitation, enabling the agent to discover optimal actions while refining its policy. Higher values of ϵ (e.g., $\epsilon = 0.5$, $\epsilon = 0.9$) caused the agent to explore excessively, reducing its ability to focus on learned strategies, while lower values hindered exploration, trapping the agent in suboptimal policies.

Number of Steps (n)

A value of $n = 5$ provided the best balance between computational efficiency and leveraging future rewards. Larger values (e.g., $n = 15$) led to instability and lower performance as the updates became delayed and extended sequence of experiences.

Performance Evaluation

The agent's performance was evaluated based on total actions and average rewards over 1000 episodes and 5 runs. The results demonstrate:

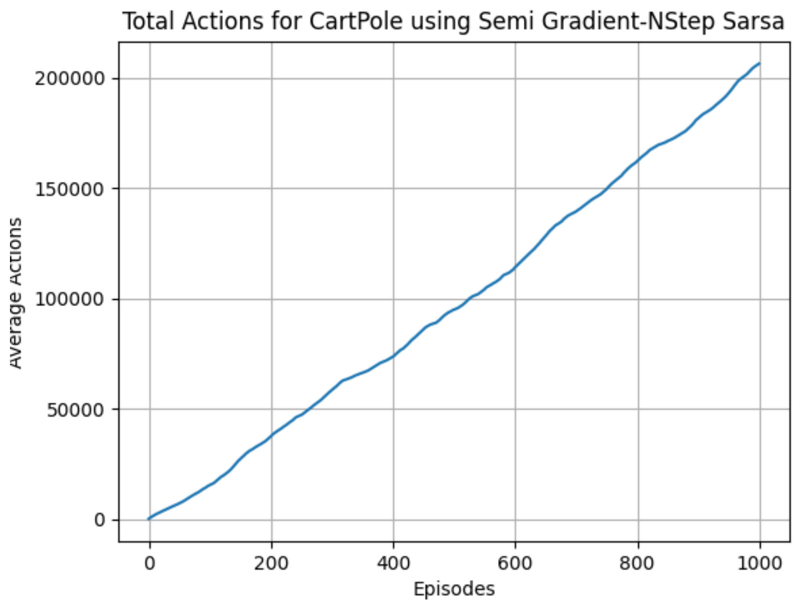
- **Total Actions:** The total actions per episode grew consistently, reaching over 150,000 by the 1000th episode. This indicates that the agent successfully learned to balance the pole for extended periods, maximizing the episode duration.
- **Average Rewards:** The mean reward improved significantly, reaching values over 400 by the end of training. The standard deviation narrowed as training progressed, highlighting stable and consistent learning.

Summary of Best Parameters

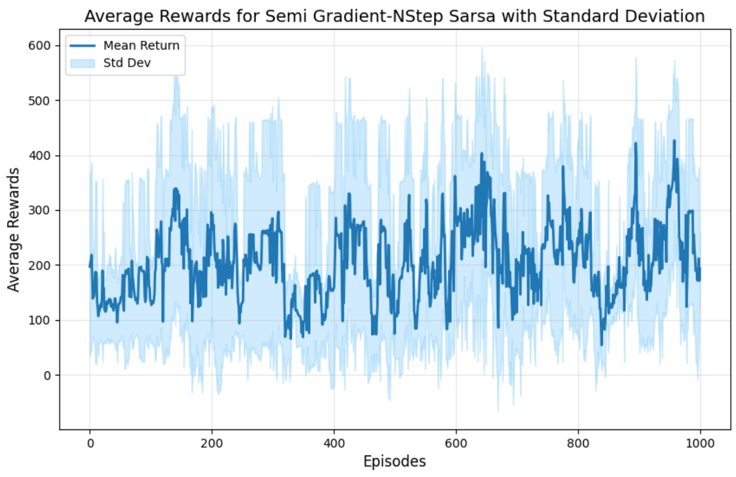
Parameter	**Optimal Value**
Hidden Layer Size	128 neurons
Discount Factor (γ)	0.99
Learning Rate (α)	0.001
Exploration Rate (ϵ)	0.1
Number of Steps (n)	5
Number of Episodes	1000
Runs	5

Table 3: Summary of Best Parameters for Semi-Gradient N-Step SARSA on CartPole

Figure 7 illustrate the training performance of the Semi-Gradient N-Step SARSA algorithm on the Cartpole environment:



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 7: Training performance of the Semi Gradient N-Step SARSA algorithm over Cartpole environment.

3.4.1 Acrobot

The Semi-Gradient N-Step Sarsa algorithm was implemented on the Acrobot environment, leveraging the distinct characteristics of this task, which involves balancing a two-link pendulum by applying discrete torques at its joints. Given the sparse reward structure and high-dimensional state space, the algorithm's hyperparameters were optimized to achieve stable learning and convergence.

The best hyperparameters were identified following similar steps taken for Cartpole environment. The final configuration comprised a hidden layer size of **128 neurons**, a discount factor (γ) of **0.99**, a learning rate (α) of **0.001**, an exploration parameter (ϵ) of **0.1**, and n-step returns with $n = \mathbf{5}$. These settings enabled the agent to achieve optimal performance, as observed over **1000 episodes** with **5 runs**.

Network Architecture: The algorithm's action-value function was parameterized using a neural network with a single hidden layer. Various configurations were tested, including smaller networks with **64 neurons** and larger networks with **256 neurons**. While the smaller network provided computational efficiency, it lacked the capacity to model the Acrobot's complex dynamics. The larger network, though more expressive, introduced instability in learning. The optimal configuration of **128 neurons** balanced capacity and stability, effectively capturing the dynamics without overfitting.

Discount Factor (γ): A discount factor of **0.99** was employed to prioritize long-term rewards, which is critical in the Acrobot environment due to its delayed reward structure. Lower values (e.g., $\gamma = 0.8$) were insufficient for effective planning, leading to suboptimal policies that failed to achieve the task's goal.

Learning Rate (α): The learning rate of **0.001** allowed stable updates to the network, enabling consistent improvement in the action-value estimates. Higher rates (e.g., $\alpha = 0.01$) caused oscillatory behavior, while lower rates (e.g., $\alpha = 0.0001$) led to excessively slow convergence.

Exploration Parameter (ϵ): An ϵ-greedy policy was adopted, with $\epsilon = \mathbf{0.1}$ striking a balance between exploration and exploitation. Higher values (e.g., $\epsilon = 0.5$) led to excessive exploration, hindering convergence, while lower values (e.g., $\epsilon = 0.01$) caused premature exploitation of suboptimal policies.

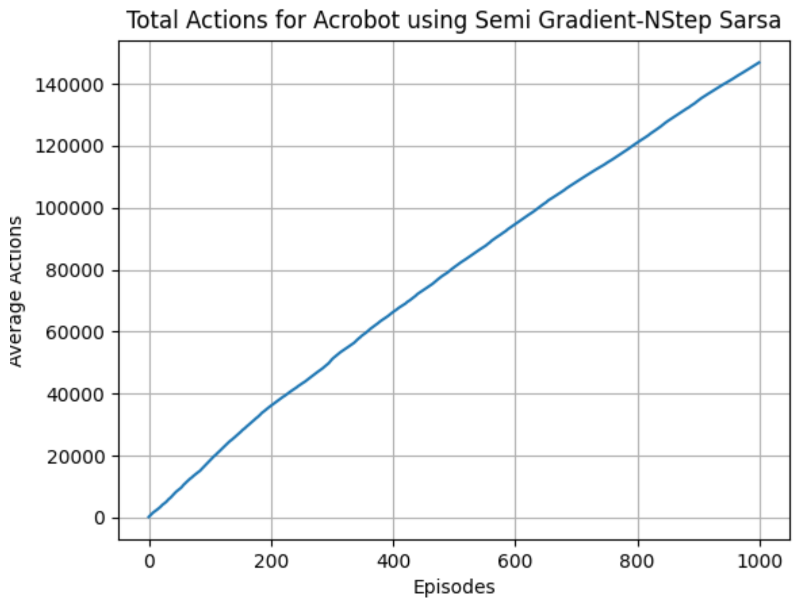
Performance Evaluation :- The agent's performance was evaluated based on total actions and average rewards over 1000 episodes and 5 runs. The results demonstrate:

- **Total Actions:** The total actions per episode grew consistently, reaching over 140,000 by the 1000th episode. This indicates that the agent successfully learned to swing up and balance the Acrobot, maximizing the episode duration and achieving the task goal.
- **Average Rewards:** The mean reward improved significantly, reaching values close to -100 by the end of training. The standard deviation narrowed as training progressed, indicating stable and consistent learning in the challenging Acrobot environment.

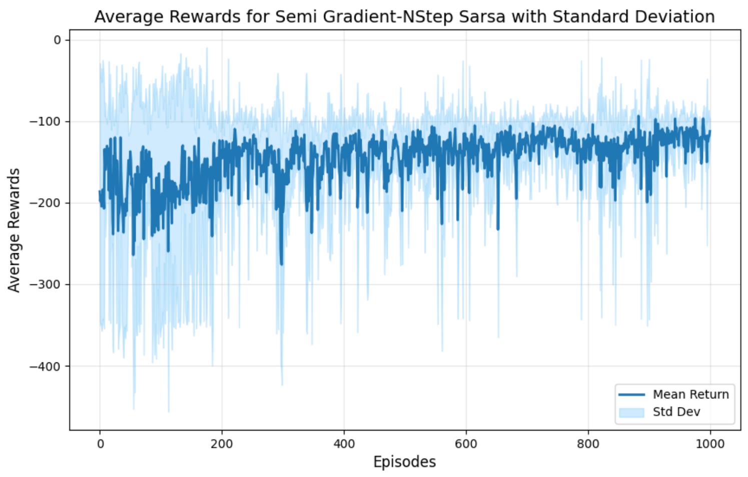
Parameter	**Optimal Value**
Hidden Layer Size	128 neurons
Discount Factor (γ)	0.99
Learning Rate (α)	0.001
Exploration Parameter (ϵ)	0.1
Number of Episodes	1000
Runs	5

Table 4: Summary of best parameters for the Semi-Gradient N-Step Sarsa algorithm on the Acrobot environment.

Summary of Best Parameters: Figure 8 illustrate the training performance of the Semi-Gradient N-Step SARSA algorithm on the Acrobot environment:



(a) Cumulative Steps Over Episodes.



(b) Mean and Standard Deviation of Rewards.

Figure 8: Training performance of the Semi Gradient N-Step SARSA algorithm over Acrobot environment.

4 Discussion and Comparison

Across the tested algorithms, **PPO consistently demonstrated superior performance**, particularly in environments with dense rewards like CartPole. It excelled by rapidly converging to optimal behaviors, achieving near-perfect scores with significantly less variance across multiple runs. However, tuning PPO's hyperparameters, including the clipping parameter (ϵ) and the learning rates ($\alpha_{\text{policy}}, \alpha_{\text{value}}$), proved to be challenging.

This complexity sometimes led to slower convergence in environments with sparse rewards, such as Acrobot.

In contrast, **REINFORCE with Baseline** displayed robust learning capabilities but required more episodes to achieve stable performance. The inclusion of a baseline helped reduce variance in policy updates, particularly in CartPole, where the algorithm learned an optimal policy after substantial training. However, in environments with sparse rewards like Acrobot, REINFORCE struggled to find optimal trajectories efficiently due to its reliance on full-episode returns.

The **One-Step Actor-Critic method** balanced policy and value updates effectively by leveraging the temporal difference (TD) error for step-by-step feedback. This made it particularly adaptive in tasks like Acrobot, where immediate feedback was crucial for learning long-term strategies. Its ability to integrate TD updates contributed to faster convergence compared to REINFORCE. However, the method was more sensitive to hyperparameter tuning, particularly the learning rates ($\alpha_{\text{policy}}, \alpha_{\text{value}}$) and discount factor ($\gamma$).

Finally, the **Semi-Gradient N-Step SARSA algorithm** performed well in environments with continuous state spaces, effectively learning policies for both CartPole and Acrobot. By leveraging multi-step returns, it achieved a balance between short-term and long-term rewards, improving stability and convergence speed. However, its dependency on the number of steps (n) required careful tuning to avoid instability.

Evaluation Metrics and Sensitivity: The algorithms were evaluated based on their ability to maximize cumulative rewards, their convergence speed, and their consistency across runs. Notably, PPO demonstrated robustness but required significant computational resources due to its complex hyperparameter tuning. Variability in performance was observed across random seeds, particularly in sparse-reward environments like Acrobot, underlining the need for careful initialization and parameter selection.

Key Observations:

1. **CartPole:** PPO and Actor-Critic achieved optimal performance faster, with PPO converging within 250 episodes. REINFORCE required significantly more episodes to stabilize, while N-Step SARSA offered competitive results with lower variance.

2. **Acrobot:** Actor-Critic outperformed other methods, effectively navigating sparse rewards and achieving the target height consistently. PPO also performed well, but its convergence was slower due to the environment's reward structure. REINFORCE exhibited higher variability, struggling with reward sparsity.

3. **Tuning Challenges:** All algorithms required extensive hyperparameter tuning, particularly PPO, where the balance between exploration and exploitation (clipping parameter ϵ) was critical. Variability across random seeds also highlighted the sensitivity of these methods.

Performance Trade-offs: While PPO demonstrated superior performance in dense-reward environments, it required careful hyperparameter tuning. Actor-Critic excelled in sparse-reward environments like Acrobot, leveraging its ability to use step-wise updates for effective learning. REINFORCE, though robust, was slower to converge due to its reliance on full-episode returns, while N-Step SARSA effectively balanced immediate and long-term feedback.

Overall, each algorithm successfully learned optimal or near-optimal behaviors, showcasing their adaptability to diverse reinforcement learning tasks. However, the choice of algorithm depended heavily on the environment's characteristics, with dense reward structures favoring PPO and sparse rewards benefiting from Actor-Critic's iterative updates.

Author Contributions: Mukta Maddipatla implemented REINFORCE with baseline and PPO algorithms. Prudhvi Nikku implemented Actor-Critic and Semi-Gradient N-Step SARSA algorithms.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

Zihao Li, Zhuoran Yang, and Mengdi Wang. Reinforcement learning with human feedback: Learning dynamic choices via pessimism. 2023.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017a. URL `http://arxiv.org/abs/1502.05477`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017b. URL `http://arxiv.org/abs/1707.06347`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004. URL `https://api.semanticscholar.org/CorpusID:19115634`.