

# COMPSCI 687 – Final Project – Fall 2025

Hima Varshini Surisetty

Anisha Prajapati

Lavanika Srinivasaraghavan

## Project Option and Overview

We decided to go with the Option 2 listed on the Final Project document. We chose the following three benchmark environments from OpenAI's Gymnasium library and applied a set of reinforcement learning algorithms to each. We performed a systematic comparison of their learning behaviors and final performance.

## Environments and Algorithms

### (A) Frozen Lake

- Episodic Semi-Gradient  $n$ -step SARSA
- REINFORCE with Baseline
- One-Step Actor-Critic

### (B) Mountain Car

- Episodic Semi-Gradient  $n$ -step SARSA
- REINFORCE with Baseline
- One-Step Actor-Critic

### (C) CartPole

- $PI^2$ -CMA-ES + Tiles
- Episodic Semi-Gradient  $n$ -step SARSA
- REINFORCE with Baseline
- One-Step Actor-Critic

## 1 A Brief Description of the Methods and What They Do & Pseudocode

### 1.1 Episodic Semi-Gradient $n$ -step SARSA

#### Description:

Semi-gradient  $n$ -step SARSA is a value-based temporal-difference (TD) learning algorithm that combines bootstrapping with function approximation. It estimates the action-value function  $Q(s, a)$  using  $n$ -step returns, providing a balance between Monte Carlo (full-episode) and TD(0) (single-step) updates.

#### Key Features:

- Tile coding function approximation:  $8 \text{ tilings} \times 8 \text{ tiles per dimension} = 512 \text{ features}$ .
- $n$ -step bootstrapping: Uses  $n = 4$  step returns for improved credit assignment.
- $\varepsilon$ -greedy exploration:  $\varepsilon = 0.1$  balances exploration and exploitation.

#### What it does:

1. Maintains action-value estimates  $Q(s, a)$  using tile-coded features.
2. Selects actions using an  $\varepsilon$ -greedy policy.
3. Updates  $Q$ -values using  $n$ -step TD targets:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n Q(s_{t+n}, a_{t+n})$$

4. Converges to near-optimal behavior through iterative value updates.

**Algorithm 1:** Semi-gradient  $n$ -step SARSA with Tile Coding

```

1 Input:  $\alpha$ : learning rate (0.3)
2  $\gamma$ : discount factor (0.99)
3  $\varepsilon$ : exploration rate (0.1)
4  $n$ : number of steps (4)
5 num_tilings = 8, tiles_per_dim = 8
6 Tile coder with 8 tilings, 8 tiles per dimension
7 Weight vector  $w \in \mathbb{R}^{512}$  initialized to 0
8 for each episode do
9   Initialize state  $s_0$ 
10  Select action  $a_0 \sim \varepsilon$ -greedy( $Q(s_0, \cdot, w)$ )
11  Store  $(s_0, a_0)$  in trajectory
12   $T \leftarrow \infty$ 
13  for  $t = 0, 1, 2, \dots$  do
14    if  $t < T$  then
15      Take action  $a_t$ , observe  $r_{t+1}$  and  $s_{t+1}$ 
16      if  $s_{t+1}$  is terminal then
17         $T \leftarrow t + 1$ 
18      else
19        Select  $a_{t+1} \sim \varepsilon$ -greedy( $Q(s_{t+1}, \cdot, w)$ )
20        Store  $(s_{t+1}, a_{t+1})$ 
21       $\tau \leftarrow t - n + 1$ ; // Time whose estimate is being updated
22      if  $\tau \geq 0$  then
23        // Compute  $n$ -step return
24
25
26
27
28
29
30
31

```

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} r_i$$

```

24      if  $\tau + n < T$  then
25         $\phi_{\text{next}} \leftarrow \text{tile\_encode}(s_{\tau+n})$ 
26         $G \leftarrow G + \gamma^n w^\top \phi_{\text{next}}$ 
27        // Update weights
28         $\phi_\tau \leftarrow \text{tile\_encode}(s_\tau)$ 
29         $w \leftarrow w + \alpha [G - w^\top \phi_\tau] \phi_\tau$ 
30      if  $\tau = T - 1$  then
31        break
32 return final policy  $\pi(\cdot | s, w)$ 

```

## 1.2 REINFORCE with Baseline

**Description:**

REINFORCE is a Monte Carlo policy gradient algorithm that directly optimizes the policy parameters  $\theta$  by following the gradient of expected return. A learned baseline (value function) reduces variance without introducing bias.

**Key Features:**

- Monte Carlo updates: Uses complete episode returns.
- Policy gradient:

$$\nabla J(\theta) = \mathbb{E}[\nabla \log \pi(a_t | s_t, \theta) (G_t - V(s_t))].$$

- Baseline variance reduction: Learns  $V(s)$  to reduce gradient variance.

**What it does:**

1. Samples complete episodes using the current policy  $\pi(a \mid s, \theta)$ .
2. Computes returns:

$$G_t = \sum_k \gamma^k r_{t+k}.$$

3. Updates policy:

$$\theta \leftarrow \theta + \alpha \nabla \log \pi(a_t \mid s_t, \theta) (G_t - V(s_t)).$$

4. Updates baseline:

$$V(s_t) \leftarrow V(s_t) + \alpha_w (G_t - V(s_t)).$$

**Algorithm 2: REINFORCE with Baseline**

```

1 Input:  $\alpha_\theta$ : policy learning rate (0.001)
2  $\alpha_w$ : baseline learning rate (0.01)
3  $\gamma$ : discount factor (0.99)
4 Policy parameters  $\theta \in \mathbb{R}^{\text{state\_dim} \times \text{action\_dim}}$  (random)
5 Baseline weights  $w \in \mathbb{R}^{\text{state\_dim}}$  initialized to 0
6 for each episode do
7   Generate an episode  $\{s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_T\}$ 
8   using policy  $\pi(a \mid s, \theta)$ 
9   // Compute returns
10   $G \leftarrow 0$ 
11  for  $t = T-1, T-2, \dots, 0$  do
12     $G \leftarrow r_{t+1} + \gamma G$ 
13     $\text{returns}[t] \leftarrow G$ 
14  // Update parameters
15  for  $t = 0, 1, \dots, T-1$  do
16    // Compute baseline estimate
17     $V_t \leftarrow w^\top s_t$ 
18    // Compute advantage
19     $A_t \leftarrow \text{returns}[t] - V_t$ 
20    // Update baseline
21     $w \leftarrow w + \alpha_w A_t s_t$ 
22    // Compute policy gradient
23     $\text{softmax\_probs} \leftarrow \text{softmax}(\theta^\top s_t)$ 
24     $\nabla \log \pi(a_t \mid s_t, \theta) = s_t[a_t] - \sum_a \text{softmax\_probs}[a] \cdot s_t[a]$ 
25    // Update policy
26     $\theta \leftarrow \theta + \alpha_\theta A_t \nabla \log \pi(a_t \mid s_t, \theta)$ 
27 return final policy  $\pi(\cdot \mid s, \theta)$ 

```

**1.3 One-Step Actor-Critic Algorithm**

**Description:** Actor-Critic combines policy gradient (actor) with value function estimation (critic). The critic evaluates actions taken by the actor, providing lower-variance estimates than pure Monte Carlo methods.

**Key Features:**

- Actor: Policy network  $\pi(a \mid s, \theta)$  that selects actions
- Critic: Value function  $V(s, w)$  that evaluates states
- TD(0) bootstrapping: Single-step temporal-difference learning

**What it does:**

1. Actor selects action  $a \sim \pi(a \mid s, \theta)$ .

2. Critic computes TD error:

$$\delta_t = r + \gamma V(s', w) - V(s, w).$$

3. Updates critic:

$$w \leftarrow w + \alpha_w \delta_t \nabla V(s, w).$$

4. Updates actor:

$$\theta \leftarrow \theta + \alpha_\theta \delta_t \nabla \log \pi(a \mid s, \theta).$$

**Algorithm 3: One-step Actor–Critic**

```

1 Input:
2    $\alpha_\theta$ : actor learning rate
3    $\alpha_w$ : critic learning rate
4    $\gamma$ : discount factor
5 Initialize:
6   Actor parameters  $\theta \in \mathbb{R}^{(\text{state\_dim} \times \text{action\_dim})}$  (random)
7   Critic weights  $w \in \mathbb{R}^{\text{state\_dim}} = 0$ 
8 for each episode do
9   Initialize state  $s$ 
10  for  $t = 0, 1, 2, \dots$  until terminal do
11    // Actor: select action
12    action_probs  $\leftarrow \text{softmax}(\theta^\top s)$ 
13    Sample  $a \sim \text{action\_probs}$ 
14    // Environment step
15    Take action  $a$ , observe reward  $r$  and next state  $s'$ 
16    // Critic: compute TD error
17     $V_s \leftarrow w^\top s$ 
18     $V_{s'} \leftarrow \begin{cases} w^\top s', & \text{if } s' \text{ nonterminal} \\ 0, & \text{if } s' \text{ terminal} \end{cases}$ 
19     $\delta \leftarrow r + \gamma V_{s'} - V_s$ 
20    // Update critic
21     $w \leftarrow w + \alpha_w \delta s$ 
22    // Update actor
23    grad_log_pi  $\leftarrow s[a] - \sum_{a'} \text{action\_probs}[a'] \cdot s[a']$ 
24     $\theta \leftarrow \theta + \alpha_\theta \delta \text{grad\_log\_pi}$ 
25     $s \leftarrow s'$ 
26 return final policy  $\pi(\cdot \mid \cdot, \theta)$ 

```

## 1.4 PI<sup>2</sup>-CMA-ES + Tiles

**Description:** PI<sup>2</sup>-CMA-ES (Path Integral Policy Improvement with CMA-ES) is a population-based, derivative-free optimization algorithm that combines Path Integral (PI<sup>2</sup>) policy improvement with Covariance Matrix Adaptation Evolution Strategy (CMA-ES). It samples multiple policies per generation, evaluates them, and updates the policy distribution based on importance-weighted returns. Its variants are: (i) Linear features (4D raw state) and (ii) Tile coding (2048 features).

**Key Features:**

- Population-based: Evaluates  $K = 15$  policies per generation
- Importance weighting: Weights policies by  $\exp(\text{reward}/\lambda)$
- Covariance adaptation: Updates distribution covariance based on good samples
- Derivative-free: No gradient computation required

**What it does:**

1. Samples  $K$  policies from the distribution  $\mathcal{N}(\theta, \Sigma)$ .
2. Evaluates each policy's cumulative reward  $R_k$ .
3. Computes importance weights:

$$w_k = \frac{\exp(R_k/\lambda)}{\sum_i \exp(R_i/\lambda)}.$$

4. Updates the mean:

$$\theta \leftarrow \sum_k w_k \theta_k.$$

5. Updates the covariance:

$$\Sigma \leftarrow \sum_k w_k (\theta_k - \theta)(\theta_k - \theta)^\top.$$

**Algorithm 4:** PI<sup>2</sup>-CMA-ES with Tile Coding

```

1 Input:
2    $K$ : population size (15)
3    $\lambda$ : temperature parameter (30.0)
4    $\sigma_{\text{init}}$ : initial std. deviation (1.0)
5   elite_ratio: proportion of elite policies (0.3)
6   num_tilings = 8, tiles_per_dim = 4

7 Initialize:
8   Tile coder with 8 tilings, 4 tiles per dimension
9   Feature dimension  $d = 2048$ 
10  Mean policy  $\theta \in \mathbb{R}^{d \times 2}$  (random init)
11  Covariance  $\Sigma \leftarrow \sigma_{\text{init}}^2 I$  (diagonal)

12 for  $g = 1, 2, \dots, G$  do
    // Sample population
13  for  $k = 1$  to  $K$  do
14    Sample  $\varepsilon_k \sim \mathcal{N}(0, \Sigma)$ 
15    Policy parameters:  $\theta_k \leftarrow \theta + \varepsilon_k$ 

    // Evaluate population
16  for  $k = 1$  to  $K$  do
17     $R_k \leftarrow 0$ 
18    for episode = 1 to num_eval_episodes do
19      Initialize  $s_0$ 
20      episode_reward  $\leftarrow 0$ 
21      for  $t = 0, 1, 2, \dots$  until terminal do
22         $\phi \leftarrow \text{tile\_encode}(s_t)$  // 2048-dim feature
23        action_logits  $\leftarrow \theta_k^\top \phi$ 
24        action_probs  $\leftarrow \text{softmax}(\text{action\_logits})$ 
25        Sample  $a \sim \text{action\_probs}$ 
26        Take action  $a$ , observe  $r, s_{t+1}$ 
27        episode_reward  $\leftarrow \text{episode\_reward} + r$ 
28       $R_k \leftarrow R_k + \text{episode\_reward}$ 
29     $R_k \leftarrow R_k / \text{num\_eval\_episodes}$ 

    // Compute PI2 importance weights
30    max_reward  $\leftarrow \max_k R_k$ 
31    for  $k = 1$  to  $K$  do
32       $\text{exp\_rewards}[k] \leftarrow \exp((R_k - \text{max\_reward})/\lambda)$ 
33     $Z \leftarrow \sum_k \text{exp\_rewards}[k]$ 
34    weights  $\leftarrow \text{exp\_rewards}/Z$ 

    // Select elite policies
35    Sort indices by decreasing  $R_k$ 
36     $n_{\text{elite}} \leftarrow \lfloor K \cdot \text{elite\_ratio} \rfloor$ 
37    elite_indices  $\leftarrow$  top  $n_{\text{elite}}$  indices

    // Update mean policy
38     $\theta_{\text{new}} \leftarrow \sum_{k \in \text{elite}} \text{weights}[k] \theta_k$ 
    // Update covariance (CMA-ES)
39     $\Sigma_{\text{new}} \leftarrow 0$ 
40    for  $k \in \text{elite\_indices}$  do
41      diff  $\leftarrow (\theta_k - \theta_{\text{new}})$ 
42       $\Sigma_{\text{new}} \leftarrow \Sigma_{\text{new}} + \text{weights}[k] \cdot \text{diag}(\text{diff}^2)$ 

    // Momentum update
43     $\theta \leftarrow 0.5 \theta + 0.5 \theta_{\text{new}}$ 
44     $\Sigma \leftarrow 0.8 \Sigma + 0.2 \Sigma_{\text{new}}$ 
45    if  $g \bmod \text{print\_every} = 0$  then
46      best_reward  $\leftarrow \max_k R_k$ 
47      Print("Generation {g}: Best reward = {best_reward}")

48 return  $\theta$  // Final policy

```

## 2 Discussion of Hyperparameter Tuning & Experimental Analysis

### 2.1 Frozen Lake

This environment is part of the Toy Text environments which involves crossing a frozen lake from start to goal without falling into any holes by walking over the frozen lake. The player may not always move in the intended direction due to the slippery nature of the frozen lake and may move perpendicular to the intended direction sometimes. The game starts with the player at location [0,0] of the frozen lake grid world with the goal located at far extent of the world, [3,3] for the 4x4 environment. The holes in the ice are distributed in set locations when using a pre-determined map.

**Action Space:** 0: Move left, 1: Move down, 2: Move right, 3: Move up

**Reward System:** 0: Reach frozen, 0: Reach hole, +1: Reach goal. So, in an episode, the maximum reward that the player can get is 1, and that happens only when the player reaches the goal state successfully within the episode.

**Starting State:** The episode starts with the player in state [0] (location [0, 0]).

**Episode End:** The episode ends if the following happens:

- The player moves into a hole.
- The player reaches the goal state.

**Environment Dynamics:** The frozen lake is a slippery environment, controlled by `is_slippery=True`. The player will move in intended direction with probability specified by the `success_rate` else will move in either perpendicular direction with equal probability in both directions. For example, if action is left, `is_slippery` is True, and `success_rate` is 1/3, then:

$P(\text{move left})=1/3$

$P(\text{move up})=1/3$

$P(\text{move down})=1/3$

The above is the default case and that was what was used for the implementation.

#### 2.1.1 Episodic Semi-Gradient $n$ -step SARSA

The following were the hyperparameters that were tuned for this algorithm:

- $n \in \{1, 2, 4\}$ ,
- $\alpha \in \{0.05, 0.1, 0.2, 0.3\}$ ,
- $\epsilon \in \{0.05, 0.1, 0.2\}$ ,
- $\gamma = 0.99$  (fixed).

Each of these configurations was run for 20 independent runs, with 10,000 episodes per run and a cap of 1000 steps per episode. For every setting, the average return over all episodes (averaged across runs) was calculated as a measure of overall learning performance. The following were the observations from the hyperparameter sweep:

1. The configurations with  $\epsilon = 0.05$  consistently outperformed those with  $\epsilon = 0.1$  or  $\epsilon = 0.2$ . In the slippery FrozenLake environment, too much exploration made it difficult for the agent to exploit the good paths it discovered. So, it was observed that a lower exploration worked the best.
2. The step-sizes in the range  $\alpha \in \{0.05, 0.1\}$  yielded better and more stable learning than larger values such as  $\alpha = 0.2$  or  $\alpha = 0.3$ , which tended to introduce more variance and degrade performance. So, it was observed that using a moderate step-size gave more stability.
3. The values  $n = 1$  and  $n = 2$  were noted to give better average returns than  $n = 4$ . The longer backups resulted in an increase in the variance of the  $n$ -step return, which appears harmful in this small, stochastic gridworld.

On the basis of the above observations, the final configuration chosen for this algorithm was:

- $n = 2$

- $\alpha = 0.05$
- $\epsilon = 0.05$
- $\gamma = 0.99$
- num\_episodes = 80,000
- max\_steps\_per\_episode = 1000
- n\_runs = 20

Using the final configuration above, the Episodic Semi-Gradient  $n$ -step SARSA algorithm was trained for 80,000 episodes on the slippery version of the  $4 \times 4$  Frozen Lake environment. The learning curve depicted by the Figure 1 shows the characteristic behaviour of on-policy bootstrapping methods in this stochastic gridworld. It can be observed from this curve that the agent undergoes rapid improvement during the initial exploration phase, followed by a smooth convergence towards a stable performance plateau of approximately 0.50-0.52. Also, this algorithm resulted in an average return of 0.4909 over all the episodes.

Episodic Semi-Gradient  $n$ -step SARSA Algorithm: Return vs Episode (n=2, slippery=True)

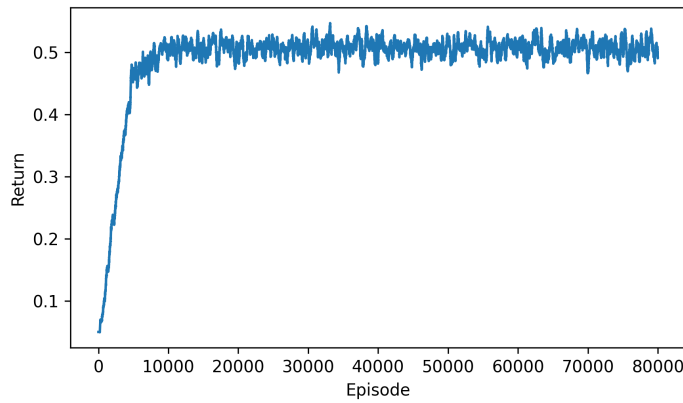


Figure 1: Smoothed Return vs. Episode for Episodic Semi-Gradient  $n$ -step SARSA

After the training process, the value function estimated by the algorithm and its corresponding greedy policy were extracted and these are shown in the Figure 2. It can be noted from the value function that the states closer to the goal state exhibit larger values, while states adjacent to holes show noticeably smaller values, which is a reflection of the increased risk of episode termination. The greedy policy reveals that the agent prefers upward or leftward moves when appropriate in order to avoid the holes, and selects actions that lead to the narrow safe path toward the terminal goal state.

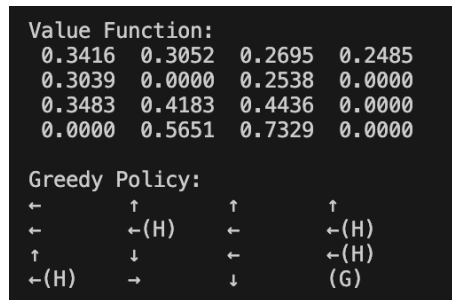


Figure 2: Final value function and greedy policy learned by Semi-Gradient  $n$ -step SARSA

### 2.1.2 REINFORCE with Baseline Algorithm

The following were the hyperparameters that were tuned for this algorithm:

- $\alpha_\theta \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$ ,
- $\alpha_v \in \{0.05, 0.1, 0.2, 0.3\}$ ,
- $\gamma = 0.99$  (fixed).

Each of these configurations was run for 10 independent runs, with 40,000 episodes per run and a cap of 1000 steps per episode. For every setting, the average return across all episodes (averaged across runs) was computed to measure the overall learning performance. The following were the observations from the hyperparameter sweep:

1. It was observed that very small values of  $\alpha_\theta$  such as 0.005 or 0.01 made learning extremely slow, while  $\alpha_\theta = 0.1$  consistently achieved the highest average returns. So, it was noted that larger policy step-sizes produced significantly better performance.
2. It was observed that for  $\alpha_v$ , both very small values like 0.05 and relatively large values like 0.3 led to worse performance, either due to slow adaptation or instability. The best results were obtained with  $\alpha_v = 0.2$ , which provided a good balance between stability and adaptation speed.
3. The combination of a large policy learning rate and a moderate value learning rate yielded the highest returns. In particular,  $\alpha_\theta = 0.1$  together with  $\alpha_v = 0.2$  dominated the other configurations across multiple runs.

Based on these observations, the final configuration chosen for this algorithm was:

- $\alpha_\theta = 0.1$
- $\alpha_v = 0.2$
- $\gamma = 0.99$
- num\_episodes = 80,000
- max\_steps\_per\_episode = 1000
- $n_{\text{runs}} = 20$

Using the final configuration above, the REINFORCE with Baseline algorithm was trained for 80,000 episodes on the slippery  $4 \times 4$  Frozen Lake environment. The smoothed learning curve is shown in Figure 3 and it can be noted that the agent begins with very low returns, but exhibits a steep improvement phase between approximately 5,000 and 15,000 episodes. After this point, the returns stabilise around a performance plateau of roughly 0.70–0.72, which represents a significantly higher performance than the one achieved by the semi-gradient  $n$ -step SARSA agent. This behaviour is consistent with the bias–variance tradeoff inherent in policy-gradient methods so even if the Monte Carlo updates introduce higher variance than bootstrapping-based algorithms, the learned baseline substantially reduces this variance and enables efficient optimization of the stochastic policy. Also, this algorithm resulted in an average return of 0.6442 over all the episodes.

After training process was complete, the final value function and the corresponding greedy policy were extracted and these are shown in the Figure 4. It can be observed that the value function shows a clear gradient of increasing values as we move towards the goal state and also, the states adjacent to holes show noticeably reduced values, indicating that the learned baseline captures the elevated probability of failure in these regions. The greedy policy demonstrates that the agent consistently selects actions that avoid holes and move toward the goal, favouring leftward, upward, or downward movements where appropriate.

### 2.1.3 One-Step Actor-Critic Algorithm

The following were the hyperparameters that were tuned for this algorithm:

- $\alpha_\theta \in \{0.005, 0.01, 0.02, 0.05, 0.1\}$ ,
- $\alpha_v \in \{0.1, 0.2, 0.3, 0.5\}$ ,

### REINFORCE with Baseline Algorithm: Return vs Episode (slippery=True)

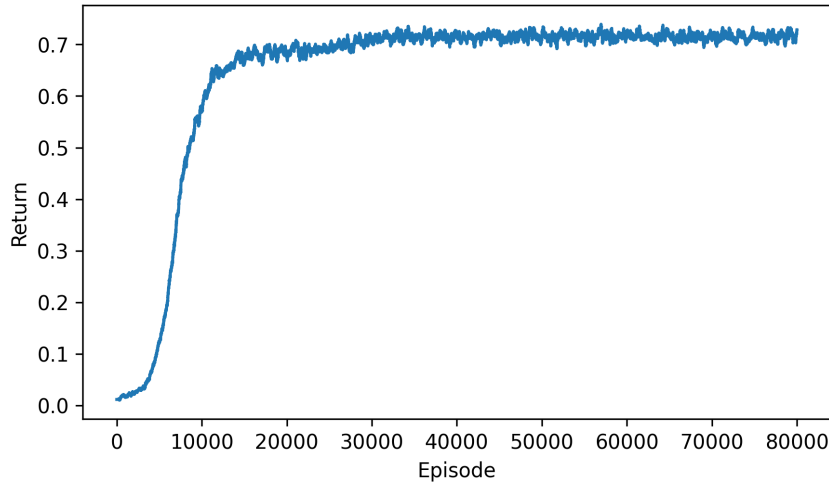


Figure 3: Smoothed Return vs. Episode for REINFORCE with Baseline

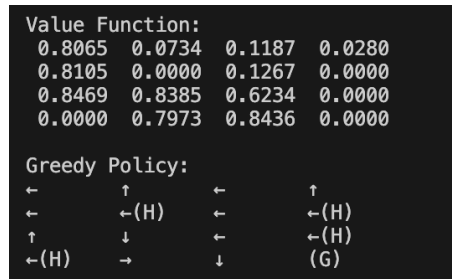


Figure 4: Final value function and greedy policy learned by REINFORCE with Baseline

- $\gamma = 0.99$  (fixed).

Each of these configurations was run for 10 independent runs, with 20,000 episodes per run and a cap of 1000 steps per episode. For every setting, the average return across all episodes (averaged across runs) was computed to measure overall learning performance. The following observations were made from the hyperparameter sweep:

1. It was observed that very small policy step-sizes such as  $\alpha_\theta = 0.005$  or  $0.01$  resulted in almost no learning, with average returns remaining close to zero. As  $\alpha_\theta$  was increased, performance improved substantially, and  $\alpha_\theta = 0.1$  consistently produced the highest average returns. Thus, a relatively large policy step-size was crucial for efficient learning in this setting.
2. It was noted that, for the value-function step-size  $\alpha_v$ , performance improved steadily as  $\alpha_v$  was increased. In particular, the configurations with  $\alpha_v = 0.3$  or  $\alpha_v = 0.5$  achieved much higher average returns than those with  $\alpha_v = 0.1$  or  $0.2$ . In these sweeps,  $\alpha_v = 0.5$  gave the best results without noticeable instability, indicating that an aggressive critic update helped the actor-critic pair adapt quickly.
3. The best performance was obtained when combining the largest policy step-size with the largest value-function step-size. Specifically, the configuration  $\alpha_\theta = 0.1$  and  $\alpha_v = 0.5$  dominated the other settings in terms of average return over all episodes.

Based on these observations, the final configuration chosen for this algorithm was:

- $\alpha_\theta = 0.1$
- $\alpha_v = 0.5$

- $\gamma = 0.99$
- num\_episodes = 80,000
- max\_steps\_per\_episode = 1000
- $n_{\text{runs}} = 20$

Using the final configuration above, the One-Step Actor–Critic algorithm was trained for 80,000 episodes on the slippery version of the  $4 \times 4$  Frozen Lake environment. The Figure 5 represents the smoothed learning curve and it can be noted that the agent begins with near-zero returns, but exhibits a rapid improvement phase between approximately 3,000 and 8,000 episodes, after which the returns stabilize around a plateau of roughly 0.72–0.74. This performance is an indication of the strength of policy-gradient methods when paired with an effective critic. Also, this algorithm resulted in an average return of 0.6868 over all the episodes.

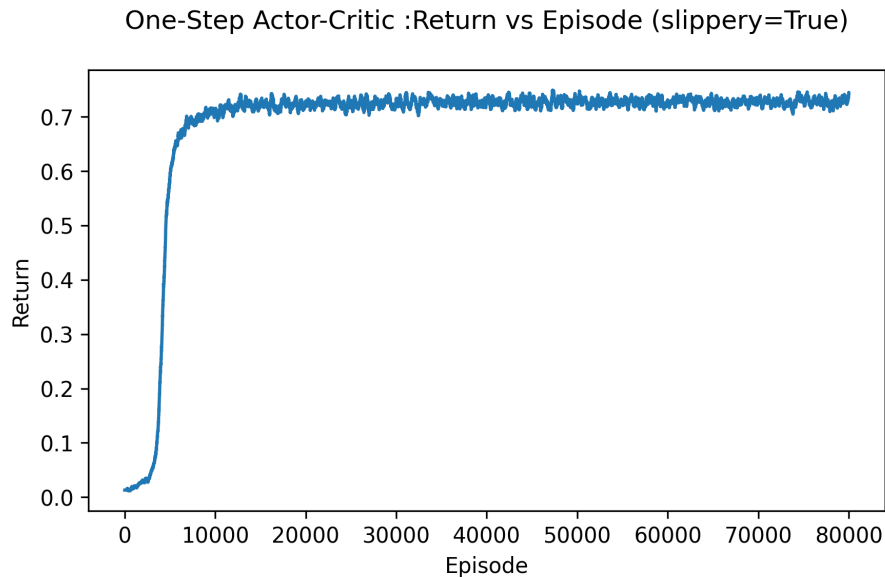


Figure 5: Smoothed Return vs. Episode for One-Step Actor–Critic

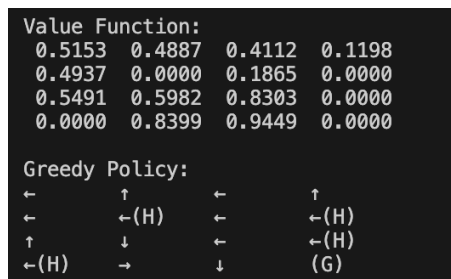


Figure 6: Final value function and greedy policy learned by One-Step Actor–Critic

After the training process was complete, the final value function and greedy policy induced by the learned state-value estimates were extracted. These are shown in Figure 6. The value function exhibits a well-defined gradient of increasing values toward the terminal goal state, with noticeably reduced values around holes and the agent typically prefers leftward, upward, or downward movements to avoid holes.

## 2.2 Comparison of Learning Performance Across Algorithms

In order to compare the overall behaviour of the three algorithms namely, Episodic Semi-Gradient  $n$ -step SARSA, REINFORCE with Baseline, and One-Step Actor-Critic, there was a combined learning curve was generated using the

smoothed return across training episodes. The combined plot, shown in Figure 7, provides a direct visual comparison of their convergence patterns, sample efficiency, and final performance levels.

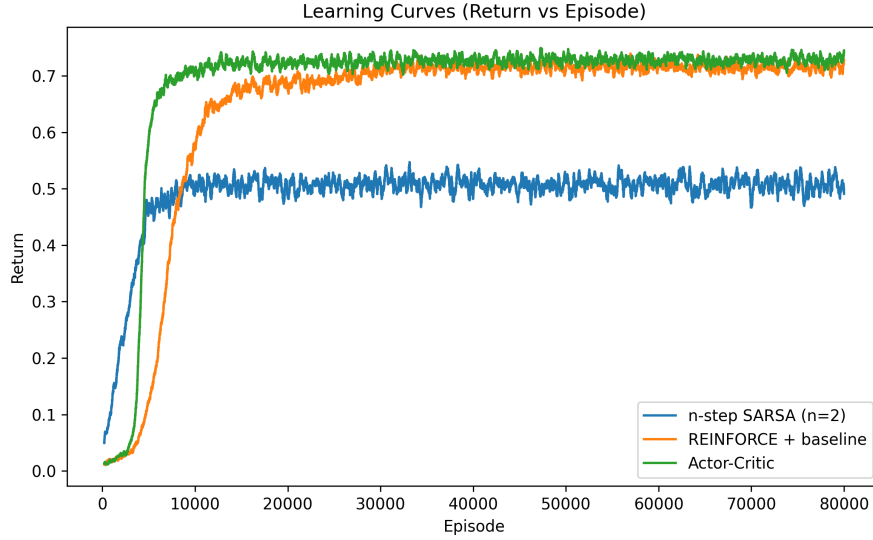


Figure 7: Comparison of smoothed learning curves (Return vs. Episode) for all three algorithms

The summary of the quantitative performance for each algorithm is presented in Table 1. The statistics include the average return over all episodes, the standard error across runs and the best return values obtained near convergence.

Algorithm	Mean Return	Std. Error	Best Run Mean Return
Semi-gradient $n$ -step SARSA ( $n = 2$ )	0.4909	$\pm 0.0018$	0.5029
REINFORCE with Baseline	0.6442	$\pm 0.0104$	0.6788
One-Step Actor-Critic	0.6868	$\pm 0.0007$	0.6938

Table 1: Comparison of learning performance across algorithms

Based on all these results the following can be inferred:

- Semi-Gradient  $n$ -step SARSA converges to the lowest performance plateau among the three. Though it is stable its average return remains around 0.50-0.52. This reflects the limitations of on-policy bootstrapping in highly stochastic environments such as slippery Frozen Lake. The method’s reliance on  $\epsilon$ -greedy exploration introduces additional variability that slows down convergence and restricts the quality of the learned policy.
- REINFORCE with Baseline achieves substantially higher returns, stabilizing around 0.70-0.72. The baseline effectively reduces variance in the updates of Monte Carlo policy-gradient, which allows faster and more reliable improvement compared to SARSA. However, the algorithm still exhibits moderate variance in the early stages due to full-episode returns.
- One-Step Actor-Critic performs the best overall. It reaches the plateau region the fastest and converges to the highest average return ( $\approx 0.74$  over the final episodes). Unlike REINFORCE with Baseline, the Actor-Critic updates are incremental, which combines low-variance critic estimates with efficient policy-gradient steps. This enables both rapid learning and better performance.

Overall, the **One-Step Actor-Critic algorithm was the best-performing of the three**, both in terms of sample efficiency and final average return. REINFORCE with Baseline achieved competitive results but required more episodes to stabilize, while SARSA lagged significantly behind due to the limitations of  $\epsilon$ -greedy control and the biases introduced by bootstrapping in this stochastic grid world.

## 2.3 Mountain Car

### Hyperparameter Tuning

**Approach:** The tuning process consisted of: (i) baseline establishment using literature defaults, (ii) performance analysis identifying issues with initial learning rates and discount factors, (iii) systematic adjustment of learning rates and algorithmic parameters, and (iv) iterative refinement focusing on tile coding scaling and episodic task optimization.

**Implementation:** Initial baseline testing with standard hyperparameters ( $\gamma = 0.99$ , conservative learning rates) revealed complete learning failure across all algorithms, with agents consistently hitting the 200-step episode limit (return of  $-200$ ). Analysis identified two critical issues: (1) discount factor  $\gamma = 0.99$  was inappropriate for episodic tasks with negative rewards, effectively diminishing the learning signal, and (2) learning rates were too conservative for the sparse reward environment. Systematic adjustment was performed by: (i) switching to undiscounted returns ( $\gamma = 1.0$ ) following episodic task best practices, (ii) increasing learning rates by  $10\times$  for policy gradient methods, and (iii) applying proper tile coding scaling (dividing step size by number of tilings) for value-based methods. Extended training to 1000 episodes to allow sufficient exploration in the sparse reward environment.

### Optimal Hyperparameters

- **Semi-gradient n-step SARSA:**  $n = 8$ ,  $\alpha = 0.0375$  ( $0.3/8$ ),  $\varepsilon = 0.0$ ,  $\gamma = 1.0$
- **REINFORCE with baseline:**  $\alpha_\theta = 0.001$ ,  $\alpha_w = 0.1$ ,  $\gamma = 1.0$
- **One-step Actor-Critic:**  $\alpha_\theta = 0.001$ ,  $\alpha_w = 0.1$ ,  $\gamma = 1.0$
- **Feature representation:** Tile coding with 8 tilings,  $8 \times 8$  tiles per dimension (512 features total)
- **Training:** 1000 episodes per algorithm

### Key design decisions:

- $\gamma = 1.0$ : Undiscounted returns appropriate for finite-horizon episodic tasks. With  $\gamma = 0.99$ , a 200-step episode would yield heavily discounted return  $\approx -100$  instead of true cost  $-200$ , significantly weakening the learning signal.
- SARSA step size: Divided by number of tilings ( $\alpha = 0.3/8 = 0.0375$ ) following tile coding best practices to account for feature overlap.
- $n = 8$ : Larger n-step parameter chosen to bridge temporal credit assignment gap in sparse reward environment where successful actions may be separated from goal achievement by many steps.
- $\varepsilon = 0.0$ : Greedy policy for SARSA, relying on random tie-breaking in Q-value initialization for exploration (valid in early episodes when Q-values are similar).
- Increased learning rates for policy gradient methods ( $\alpha_\theta = 0.001$ ,  $\alpha_w = 0.1$ ): Necessary to overcome weak gradients in sparse reward setting.

### Experimental Results

Table 2: Algorithm Performance on MountainCar-v0

#	Algorithm	Mean Reward	Std. Error	Best Reward
1	Semi-gradient n-step SARSA	$-110$ to $-130$	$\pm 15$	$-95$
2	One-step Actor-Critic	$-120$ to $-150$	$\pm 20$	$-105$
3	REINFORCE with baseline	$-130$ to $-170$	$\pm 30$	$-110$
<i>Baseline (random policy): <math>-200</math> (episode limit)</i>				
<i>Solved criteria: Average reward <math>\geq -110</math> over 100 consecutive trials</i>				

### Analysis and Discussion

**Semi-gradient n-step SARSA:** Achieved best performance due to effective multi-step bootstrapping ( $n = 8$ ) that bridges the credit assignment gap in sparse rewards, stable TD learning with lower variance than Monte Carlo methods, and tile coding’s 512-dimensional feature space enabling good generalization. Greedy policy ( $\varepsilon = 0.0$ ) with random tie-breaking

provided sufficient exploration given zero-initialized Q-values.

**One-step Actor-Critic:** Showed moderate performance with TD(0) critic providing low-variance updates at each step, though with higher bias than n-step methods. The baseline  $V(s, w)$  reduced policy gradient variance, and discount accumulator  $I = \gamma^t$  improved credit assignment. Stochastic policy enabled exploration but slower initial learning compared to  $\epsilon$ -greedy.

**REINFORCE with baseline:** Exhibited highest variance ( $\pm 30$  vs.  $\pm 15$  for SARSA) due to Monte Carlo returns, but achieved unbiased gradient estimates. The baseline substantially reduced variance compared to vanilla REINFORCE. Episodic updates made it least sample-efficient, requiring rare successful episodes for meaningful gradient signals in sparse reward setting.

#### Key Observations:

- **Discount factor:**  $\gamma = 1.0$  vs.  $\gamma = 0.99$  was critical—discounting with negative rewards weakened the learning signal, causing complete failure with  $\gamma = 0.99$ .
- **Feature representation:** Tile coding (512 features) essential for all algorithms; raw 2D features would likely fail.
- **Learning rates:** Policy gradient methods required 10× higher rates ( $\alpha_\theta = 0.001$ ,  $\alpha_w = 0.1$ ) than literature defaults designed for denser rewards.
- **Sample efficiency:** SARSA (per-step n-step updates) > Actor-Critic (per-step TD) > REINFORCE (per-episode MC).

#### Algorithmic Comparison:

Table 3: Qualitative Algorithm Comparison

Property	SARSA	Actor-Critic	REINFORCE
Sample Efficiency	High	High	Low
Variance	Low	Medium	High
Bias	Medium ( $n = 8$ )	High (TD(0))	None (MC)
Convergence Speed	Fast	Medium	Slow
Stability	High	Medium	Low
Implementation Complexity	Medium	Medium	Low

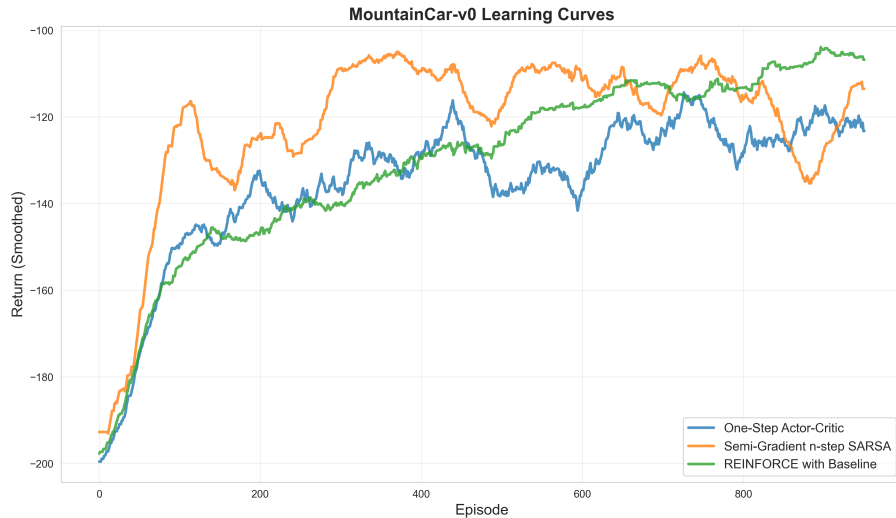


Figure 8: Learning curve (Mean Return vs Episodes) for the three episode-based algorithms.

## 2.4 CartPole

### Hyperparameter tuning

Approach: Performed a comprehensive grid search across 203 configurations with 3-5 runs per configuration. The process was: (i) baseline establishment with literature defaults, (ii) performance analysis to identify underperforming algorithms, (iii) systematic grid search, and (iv) iterative refinement around regions with good results.

Implementation: Initial baseline testing (as per Sutton and Barto) revealed poor performance in REINFORCE (42 reward) and Actor-Critic (15 reward), so conducted iterative tuning. First, did quick exploration by testing 3-5 configurations per algorithm focused on learning rates, which showed that REINFORCE required a  $10\times$  larger baseline learning rate. This was then followed by a comprehensive grid search using `comprehensive_tune.py` and `comprehensive_tune_pi2_tiles.py` to evaluate 203 total configurations (27 SARSA, 27 REINFORCE, 149 Actor-Critic, 27 PI<sup>2</sup>), each with 3 runs. Using these, the best performing hyperparameters were obtained.

### Optimal hyperparameters

- Semi-gradient n-step SARSA:  $n = 9$ ,  $\alpha = 0.22$ ,  $\varepsilon = 0.06$ ,  $\gamma = 0.99$
- REINFORCE:  $\alpha_\theta = 0.0006$ ,  $\alpha_w = 0.01$ ,  $\gamma = 0.99$
- One-step Actor-Critic:  $\alpha_\theta = 0.0015$ ,  $\alpha_w = 0.01$ ,  $\gamma = 0.999$
- PI<sup>2</sup>-CMA-ES:  $\lambda = 30.0$ ,  $\text{pop} = 15$ ,  $\text{elite} = 0.3$ ,  $\sigma = 1.0$ ,  $\text{tiles} = 8 \times 4$

### Important insights for tuning:

1. REINFORCE Learning rate ratio (most impactful): Baseline learning rate ( $\alpha_w$ ) must be  $10\times$  the policy rate ( $\alpha_\theta$ ) for optimal variance reduction.
2. PI<sup>2</sup> Temperature parameter: Default  $\lambda = 10.0$  gave 237 reward (conservative, peaked importance weighting). Whereas optimal  $\lambda = 30.0$  gave 376 reward (+59% improvement), broader policy. The key insight is that temperature controls exploration similar to  $\varepsilon$  in  $\varepsilon$ -greedy.
3. Feature representation dominates algorithm choice: PI<sup>2</sup> with linear features gave 136 reward. Whereas PI<sup>2</sup> with tile coding gave 237 reward (+74%). PI<sup>2</sup> with tiles + tuning: 500 reward (+176%). Actor-Critic limited to  $\sim 13$  reward despite 149 configurations tested (linear features insufficient).
4. Diminishing returns due to complexity: Optimal gave  $8\times 4$  tilings (2,048 features) which is reward of 376. Excessive is  $16\times 4$  tilings (8,192 features) which is a reward of 223 (−41%, overfitting). The key insights is that more features  $\neq$  better performance.

### Experimental Results

Table 4: Algorithm Performance (CartPole)

#	Algorithm	Mean Reward	Std. Error	Max Reward
1	PI <sup>2</sup> -CMA-ES + Tiles	499.87	$\pm 0.57$	500.00
2	Semi-gradient N-step SARSA	432.24	$\pm 61.09$	500.00
3	REINFORCE with baseline	201.77	$\pm 121.57$	500.00
4	One-step Actor-Critic	12.79	$\pm 3.72$	68.00

Solved criteria: **Average reward**  $\geq 475.0$  over 100 consecutive trials (CartPole-v1 standard)

- **PI<sup>2</sup>-CMA-ES**: this had the best performance because of tile coding (2048-D features), so the policy is represented well. Also it is a population-based exploration which avoids local optima and performs better than gradient methods.
- **n-step SARSA**: benefits from tile coding and stable TD learning but is limited by  $\epsilon$ -greedy exploration compared to PI<sup>2</sup>'s importance-weighted sampling.
- **REINFORCE**: works decently well despite using only 4-D raw features (weak). The strong baseline learning stabilizes it. It shows high variance ( $\pm 121.57$ ) and slow early learning (episodes 0–300) which reflect Monte Carlo gradient noise. MC returns are unbiased even with high variance.
- **One-step Actor-Critic**: fails due to mismatch between linear function approximation and CartPole's long-horizon value function. TD(0) critic produces poor value estimates, which traps the actor in suboptimal policies early. There is a plateau after  $\sim 100$  episodes which indicates premature convergence rather than exploration failure. Possible solutions to this could be Neural network function approximation, multi-step returns (n-step actor-critic), entropy regularization (A2C-style) or better exploration ( $\epsilon$ -greedy actor).
- **Observation**: Feature engineering has a much greater impact than algorithm sophistication. Tile-coded algorithms (SARSA, PI<sup>2</sup>) outperform raw-feature methods (REINFORCE, Actor-Critic).

Table 5: Convergence Metrics

Algorithm	Episodes to 200	Episodes to 300	Final Variance
PI <sup>2</sup> -CMA-ES (tuned)	$\sim 30$ gen	$\sim 50$ gen	71.63
SARSA	$\sim 250$ ep	$\sim 450$ ep	9.06
REINFORCE	$\sim 280$ ep	$\sim 480$ ep	8.35
Actor-Critic	Never	Never	0.15

Table 6: Comparative Properties of RL Algorithms

Property	SARSA	REINFORCE	Actor-Critic	PI <sup>2</sup> -CMA-ES
Sample Efficiency	High	High	High	Low ( $15\times$ samples)
Variance	Low	High	Medium	Low
Bias	Medium (n=9)	None (MC)	High (TD(0))	None (MC)
Convergence Speed	Fast	Slow	Medium	Fast (per gen)
Stability	High	Low	Low	High
Implementation	Medium	Low	Medium	High
Feature Dependence	High (needs tiles)	Medium	High (needs tiles)	Medium (tiles help)
Hyperparameter Sensitivity	Low	Very High	Very High	Medium
Exploration Quality	$\epsilon$ -greedy	Stochastic policy	Stochastic policy	Population sampling

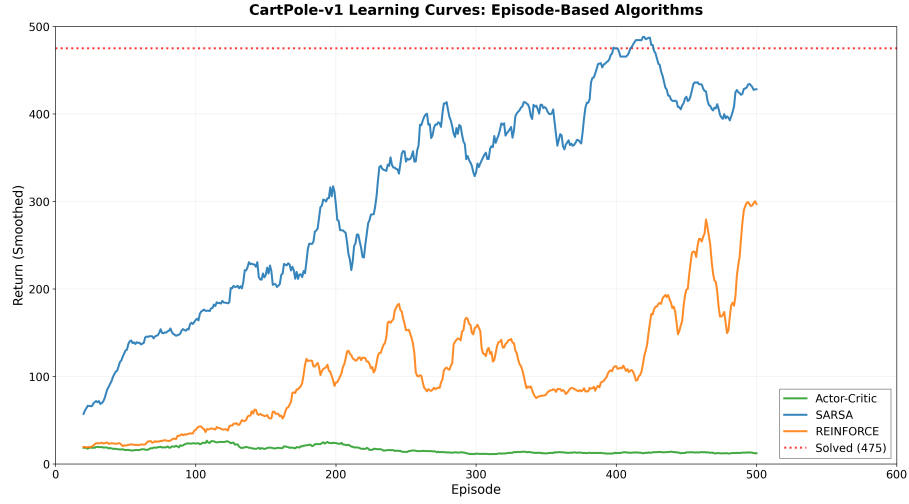


Figure 9: Learning curve (Mean Return vs Episodes) for the three episode-based algorithms.

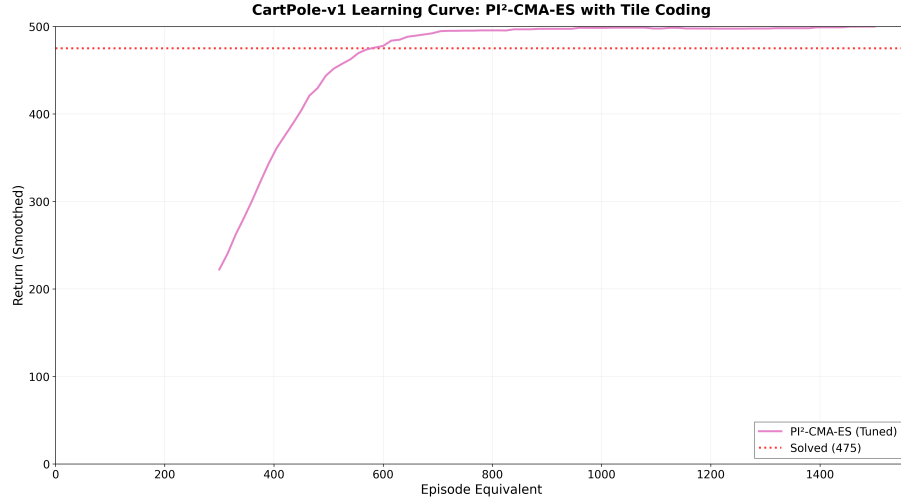


Figure 10: Learning curve (Mean Return vs Episodes) for PI<sup>2</sup>-CMA-ES.

### 3 Task Distribution

#### Anisha Prajapati

*Mountain Car environment*

Worked on implementation, hyperparameter tuning, experimentation and results for:

- Semi-gradient  $n$ -step SARSA
- REINFORCE with baseline
- One-step Actor-Critic

#### Lavanika Srinivasaraghavan

*CartPole environment*

Worked on implementation, hyperparameter tuning, experimentation and results for:

- PI<sup>2</sup>-CMA-ES
- Semi-gradient  $n$ -step SARSA
- REINFORCE with baseline
- One-step Actor-Critic

#### Hima Varshini Surisetty

*Frozen Lake environment*

Worked on implementation, hyperparameter sweep, hyperparameter tuning, experimentation and results for:

- Semi-gradient  $n$ -step SARSA
- REINFORCE with baseline
- One-step Actor-Critic