PROJECT REPORT

1 Introduction

In this project, we have implemented One-step Actor-Critic and Episodic Semi-Gradient n-step SARSA algorithms. We utilize existing MDPs such as Cartpole and Acrobot from Gym, as well as the 687-Gridworld.

2 Environments

2.1 Acrobot

The Acrobot environment is based on Sutton's work in "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding" and Sutton and Barto's book. The system consists of two links connected linearly to form a chain, with one end of the chain fixed. The joint between the two links is actuated. The goal is to apply torques on the actuated joint to swing the free end of the linear chain above a given height while starting from the initial state of hanging downwards.

For this project, we selected the "Acrobot-v1" version from the OpenAI Gym.

State: $s = (\theta_1, \theta_2,$ Angular velocity of $\theta_1,$ Angular velocity of $\theta_2)$, where:

- θ_1 represents the angle of the first joint. An angle of 0 indicates the first link is pointing directly downwards.
- θ_2 is relative to the angle of the first link. An angle of 0 corresponds to having the same angle between the two links.

Actions: $a \in \{0, 1, 2\}$. The action denotes the torque applied on the actuated joint between the two links.

Rewards: $R_t = -1$ always, except when transitioning to s_∞ (from s_∞ or from a terminal state), in which case $R_t = 0$. The goal is to have the free end reach a designated target height in as few steps as possible, and as such all steps that do not reach the goal incur a reward of -1.

Terminal States: The episode terminates when one of the following occurs, the free end reaches the target height, which is constructed as: $-\cos(\theta_1) - \cos(\theta_2 + \theta_1) > 1.0$ or the episode length is greater than 500.

Dynamics: The next state is determined using the Gym library.

2.2 CartPole

CartPole classical MDP, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

For this project, we selected the "CartPole-v1" version from the OpenAI Gym.

State: $s = (x, v, \theta, \dot{\theta})$, where:

- x denotes cart position,
- v denotes cart velocity,
- θ denotes pole angle,
- $\dot{\theta}$ denotes pole angular velocity.

with the following ranges:

$$x \in [-4.8, 4.8],$$
$$v \in [-4, 4],$$
$$\theta \in [-0.418, 0.418],$$
$$\dot{\theta} \in [-2.5, 2.5].$$

For v and $\dot{\theta}$, these are the ranges which was interpolated

Actions: $a \in \{0, 1\}$. The action indicates the direction of the fixed force applied to the cart.

Rewards: $R_t = -1$ always, except when transitioning to s_∞ (from s_∞ or from a terminal state), in which case $R_t = 0$.

Terminal States:

The episode endswhen one of the following occurs,

1. Pole Angle > $\pm 12^\circ$
2. Cart Position > ± 2.4
3. Episode Length > 500

Dynamics: The next state is determined using the Gym library.

2.3 687-GridWorld

The GridWorld environment consists of a grid with 23 states arranged in a 5x5 layout.24th state is S infinity

State Space: The state space consists of discrete states represented as cells in the grid. Each state corresponds to a specific cell in the grid.

Actions: Attempt Up (AU), Attempt Down (AD), Attempt Left (AL), Attempt Right (AR).

Dynamics: The robot's movement dynamics involve probabilistic outcomes based on the attempted action:

- With a probability of 0.8, the robot moves in the specified direction.
- With a probability of 0.05, the robot gets confused and veers to the right, resulting in a $+90^\circ$ turn from the intended direction.
- With a probability of 0.05, the robot gets confused and veers to the left, resulting in a -90° turn from the intended direction.
- With a probability of 0.1, the robot temporarily breaks and does not move.

If the movement defined by these dynamics would cause the agent to exit the grid or hit an obstacle, the agent does not move. The robot starts in state 1, and the process ends when the robot reaches state 23. Note that the robot does not have a specific facing direction, only a position indicated by the state number.

Rewards:

- Entering the state with water (state 21) yields a reward of -10. If the agent remains in state 21 due to hitting a wall or temporary breaking, it counts as "entering" the water state again and results in an additional reward of -10.
- Entering the goal state (state 23) yields a reward of +10.
- Entering any other state results in a reward of zero.

A reward discount parameter $\gamma = 0.9$ is used, which plays a role in future rewards' computation.

3 Episodic Semi-Gradient n-step SARSA

Episodic Semi-Gradient n-step SARSA is a reinforcement learning algorithm that combines elements of both TD learning and n-step methods to update Q-values. The "n" in n-step SARSA refers to the number of steps the algorithm looks ahead to update Q-values. Instead of updating Q-values after every action, n-step SARSA waits for "n" steps to elapse before updating.

The fundamental concept behind n-step SARSA involved initiating from a random initial state and choosing an epsilon-greedy action to transition to the subsequent state. Every state, action, and reward encountered was stored in separate lists.

The weights are updated in this algorithm at each time step by utilizing the temporal difference, which is the difference between the n-step return of the state-action pair (S_t, A_t) and the value estimated from the parameterized value function $\hat{q}(S_t, A_t, w)$ at $\tau = t - n + 1$.

Pseudo Code:

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 $\varepsilon > 0$, a positive integer n
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t, A_t, and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    If $t < T$, then:
      Take action A_t
      Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
      If S_{t+1} is terminal, then:
        $T \leftarrow t + 1$
      else:
        Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$ $(G_{\tau:\tau+n})$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w})\right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$
  Until $\tau = T - 1$

Figure 1: Episodic Semi-Gradient n-step SARSA Pseudocode

3.1 Cart Pole

1. Used epsilon greedy to choose action with $\epsilon = 0.9$

2. *n*-steps : Experimenting with $n - steps \in [1, 2, 4, 8]$ .As seen in the graph *n*-steps = 1 is not getting reward more that 0. and *n*-steps = 2 4 and 8 the reward gave good results.

3. Learning Rate (α): α_θ aids in learning the policy approximation function. Higher values lead to faster convergence but might cause high jitters or variations or no learning at all . Lower values prevent these jitters but significantly increase convergence time. Experimenting with $\alpha_\theta \in [0.0001, 0.001, 0.01, 1]$. As seen in the graph $\alpha_\theta = 1$ and 0.1 is not getting reward more that 0. and $\alpha_\theta = 0.01$ the rward is not reaching anothing more than 300. $\alpha_\theta = 0.001$ gave best results.

4. Feature Representation: Fourier series with Cosine.

5. Order: Increasing feature *order* expands the feature representation, giving better state representation but also increase time complexities. Experimenting with $order \in [1, 4]$ and , for $order \in [3, 4]$, the agent reaches the optimal value (500) in fewer episodes. For $order \in [1, 2]$, it reaches it in around 300 to 400 episodes. . Choosing $order = 2$ as it best optimal fit in terms of faster run time and faster convergence (reaching optimal reward of 500) at around 350 episodes.

6. Discount Factor (λ): Used $\lambda = 1$ as specified in the OpenGym Cartpole MDP.

7. Optimal Hyperparameters: $\alpha = 0.001$, $order = 2$,*n*-steps = 8,$\epsilon = 0.9$, episodes = 800 .

The graphs are based on $\alpha = 0.001$, order = 2, $\lambda = 1$, $\epsilon = 0.9$, episodes = 500, n-steps = 8, and decaying ϵ by 0.1 for every 50 episodes.

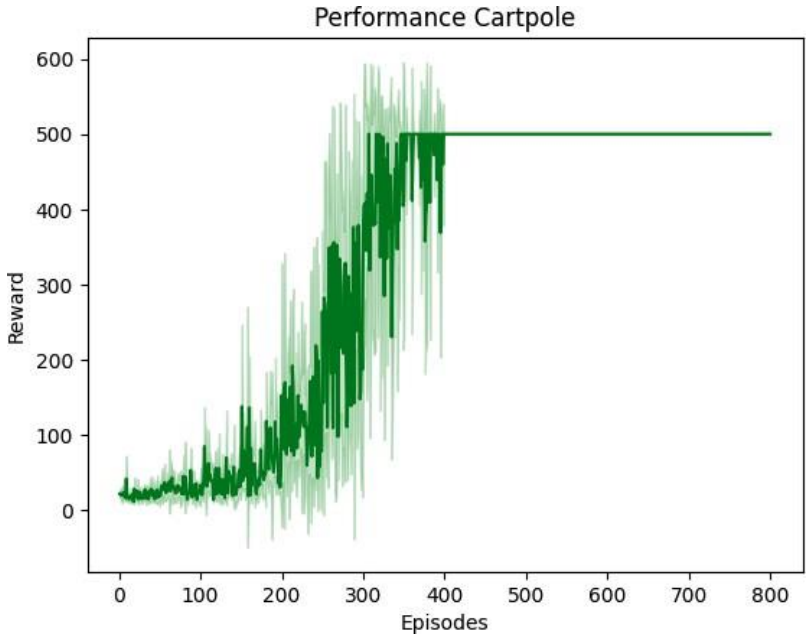

Figure 2: Reward for Cartpole

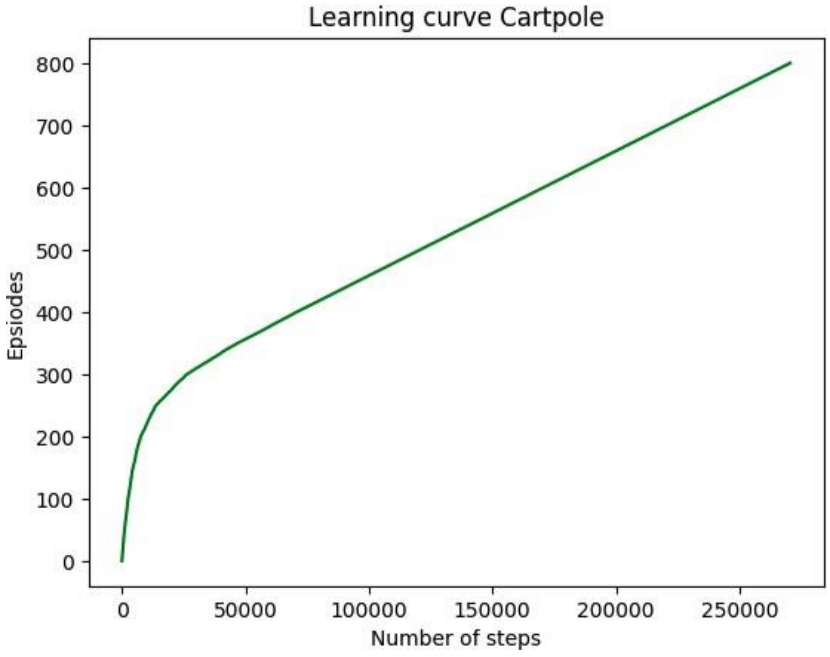

Figure 3: Learning curve for Cartpole

The graphs below shows the value of the reward function with varying the values of alpha.

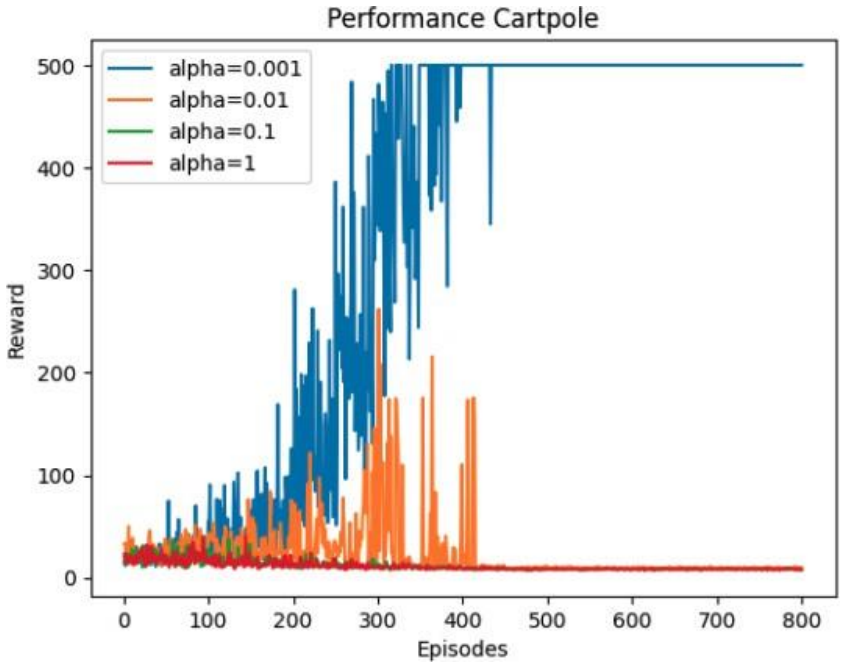

Figure 4: Reward for cartpole with varying alphas.

The graph below shows Learning curve with varying the values of alpha.

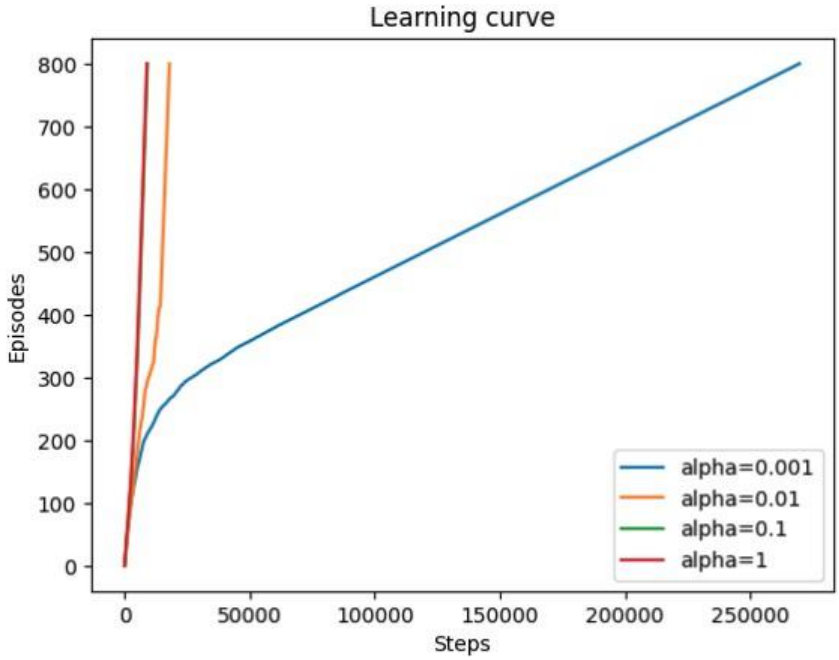

Figure 5: Learning curve for cartpole with varying alphas.

The graph below shows the value of the reward with varying values of *n*-steps.

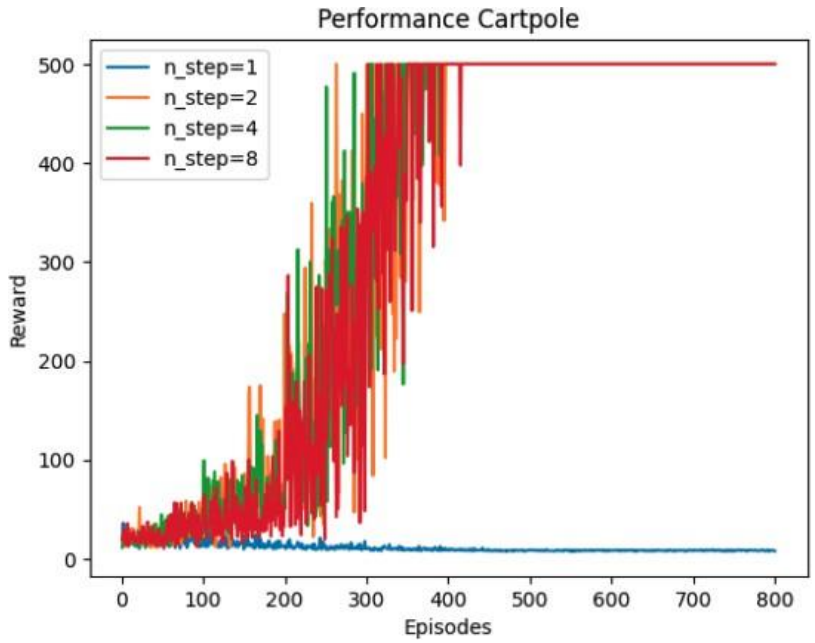

Figure 6: Reward for cartpole with varying *n-steps*

The graph below shows Learning curve with varying values of *n*-steps.

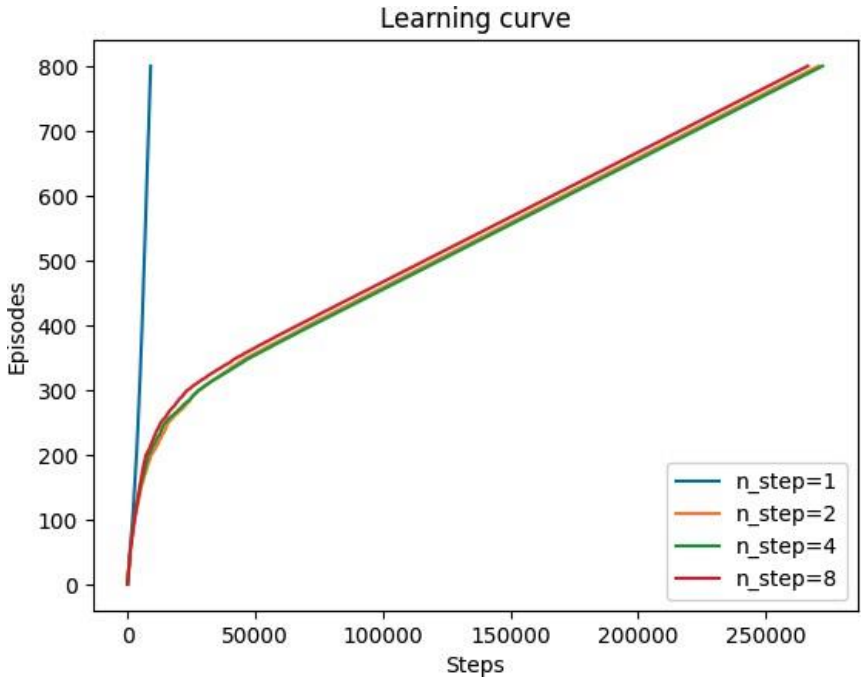

Figure 7: Learning curve cartpole with varying *n-steps*

3.2 Acrobot

1. Used epsilon greedy to choose action with $\epsilon = 0.9$
2. n-steps : Experimenting with $n - steps \in [1, 2, 4, 8]$. As seen in the graph n-steps = 1 and 2 is not getting reward more that 0. and n-steps = 4 and 8 the reward gave good results. With n-steps giving the most optimal rewards.
3. Learning Rate (α): α_θ aids in learning the policy approximation function. Higher values lead to faster convergence but might cause high jitters or variations or no learning at all . Lower values prevent these jitters but significantly increase convergence time. Experimenting with $\alpha_\theta \in [0.0001, 1]$. As seen in the graph $\alpha_\theta = 1$ and 0.1 is not getting reward more that -500. and $\alpha_\theta = 0.01$ and 0.001 the curve gotten is good in the alpha varying graph. But still $\alpha_\theta = 0.001$ gave best results.
4. Feature Representation: Used Fourier series with Cosine.
5. Order: Increasing n expands the feature representation, giving better state representation but also increase time complexities. Experimenting with $order \in [1, 4]$ and , for $order \in [3, 4]$, the agent reaches the near optimal value (-100) in fewer episodes. For $order \in [1, 2]$, it reaches it in around 200 to 300 episodes. . Choosing $order = 1$ as the state are already a lot and increasing order will increase time taken to run the algorithm and not increasing the the optimal value output as much as expected for the disadvantage wrt to time taken.
6. Discount Factor (λ): Used $\lambda = 1$ as specified in the OpenGym Acrobot MDP.
7. Optimal Hyperparameters: $\alpha = 0.001$, order = 1, $\epsilon = 0.1$, episodes = 1000, n-steps = 8.

The graphs are based on $\alpha = 0.001$, order = 1, $\lambda = 1$, $\epsilon = 0.1$, episodes = 1000, n-steps = 8.

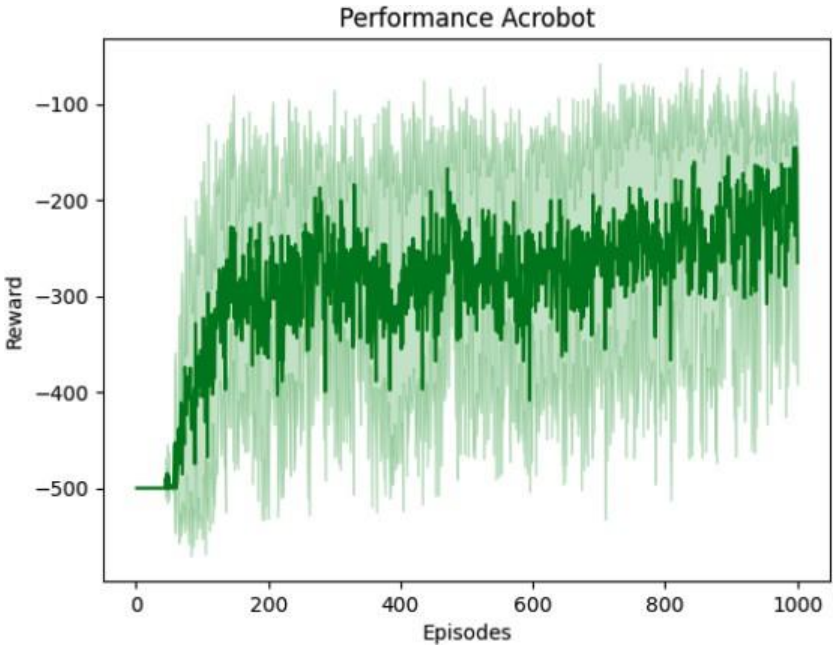

Figure 8: Reward for Acrobot

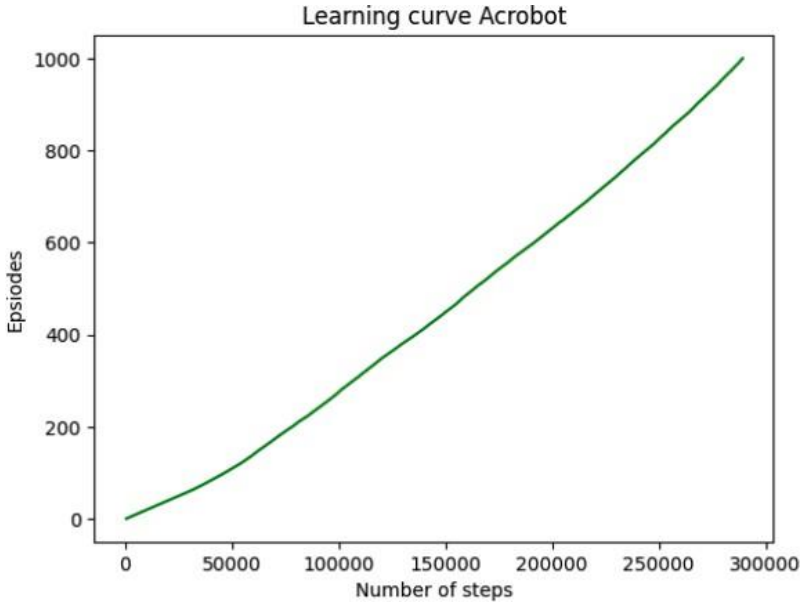

Figure 9: Learning curve for Acrobot

The graph below shows the value of the reward function with varying the values of alpha.

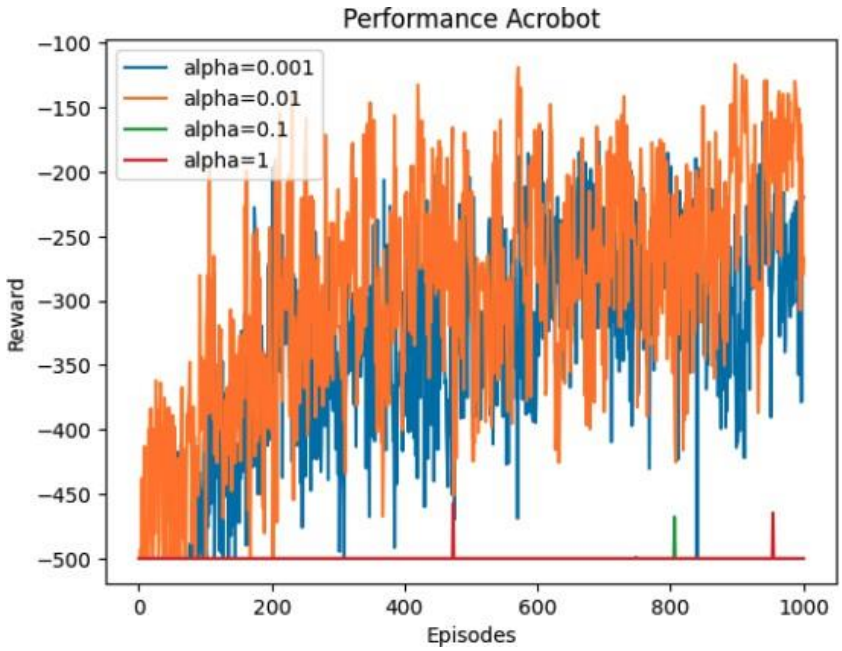

Figure 10: Reward for Acrobot with varying alphas.

The graph below shows LearningCurve with varying the values of alpha.

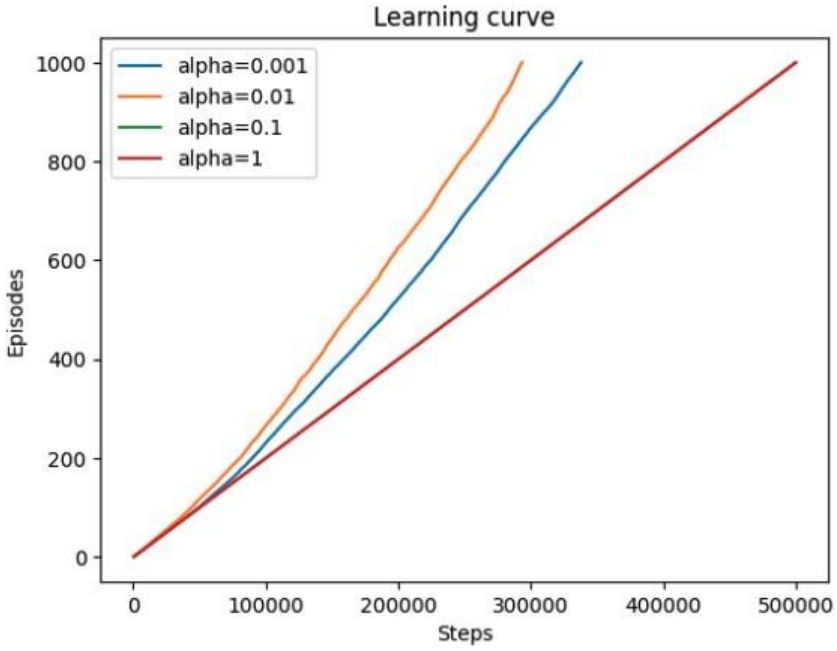

Figure 11: Learning curve for Acrobot with varying alphas.

The graph below shows the value of the reward with varying values of *n*-step.

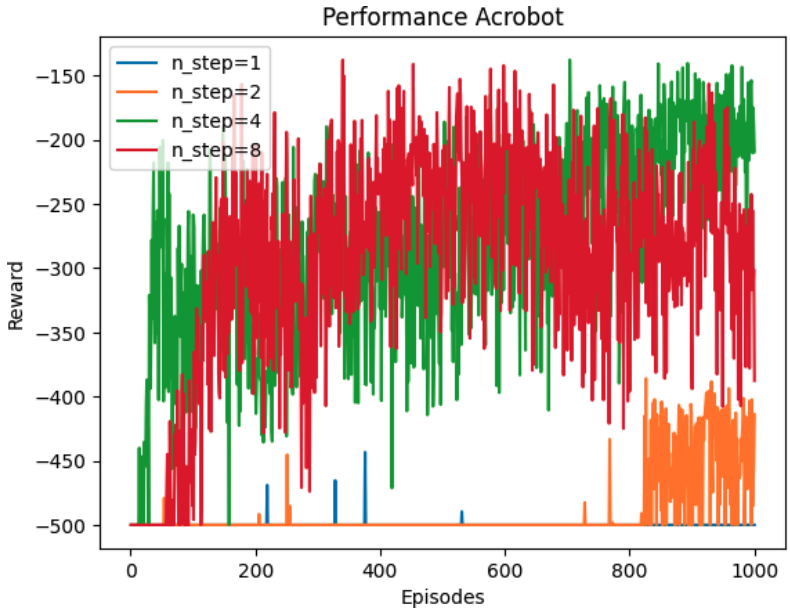

Figure 12: Reward for Acrobot with varying *n_steps*

The graph below shows Learning Curve with varying values of *n*-step.

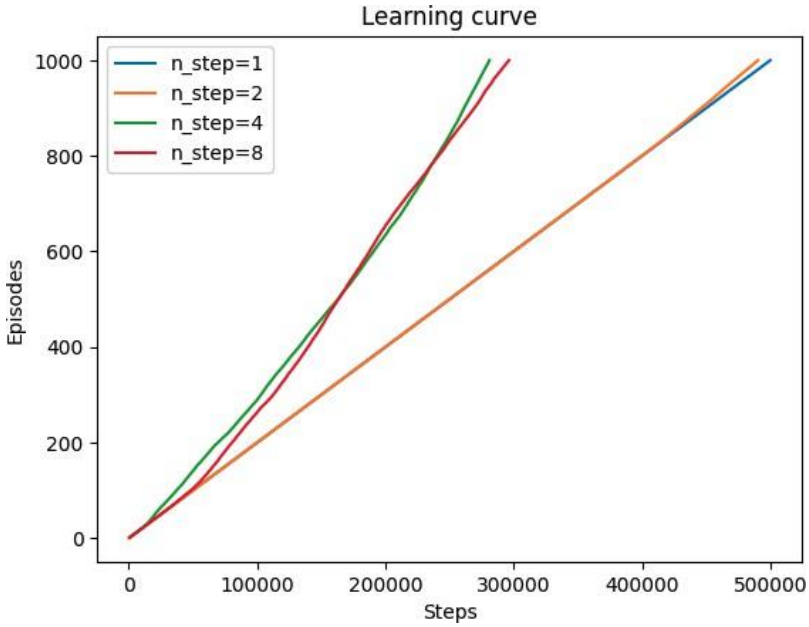

Figure 13: Learning curve for Acrobot with varying *n_steps*

3.3 GridWorld

1. Used epsilon greedy to choose action with $\epsilon = 0.9$
2. n-steps: Experimenting with n-steps $\in [1, 2, 4, 8]$.
   As observed in the graph, n-steps = 1 and 2 did not yield rewards greater than 0, while n-steps = 4 and 8 provided good results. Among these, n-steps gave the most optimal rewards.
3. Discount Parameter (λ): Used discount parameter $\lambda = 0.9$ as specified in the MDP.
4. Learning Rate (α): α_θ aids in learning the policy approximation function. Higher values lead to faster convergence but might cause high jitters or variations, or no learning at all. Lower values prevent these jitters but significantly increase convergence time. Experimenting with $\alpha_\theta \in [0.0001, 0.01]$.
   As seen in the graph, $\alpha_\theta = 0.01$ caused the cumulative reward graph to behave unexpectedly with no clear learning curve. However, $\alpha_\theta = 0.001$ gave the best results.
5. Feature Representation: Used Fourier series with Cosine.
6. Order: As Gridworld has discrete states, capturing higher order complexities of states is not required since the states are discrete. There aren't many complexities to capture. Hence, order 1 was used.
7. Discount Factor (λ): Used $\lambda = 0.9$ as specified Gridworld MDP.
8. Optimal Hyperparameters: $\alpha = 0.001$, order = 1, $\epsilon = 0.9$, episodes = 1000, n-steps = 1.

The curves are based on $\alpha_w = 0.001$, features$_n = 2$, $\lambda = 1$, $\epsilon = 0.9$, episodes = 500, n-steps = 8, and decaying ϵ by 0.1 for every 50 episodes.

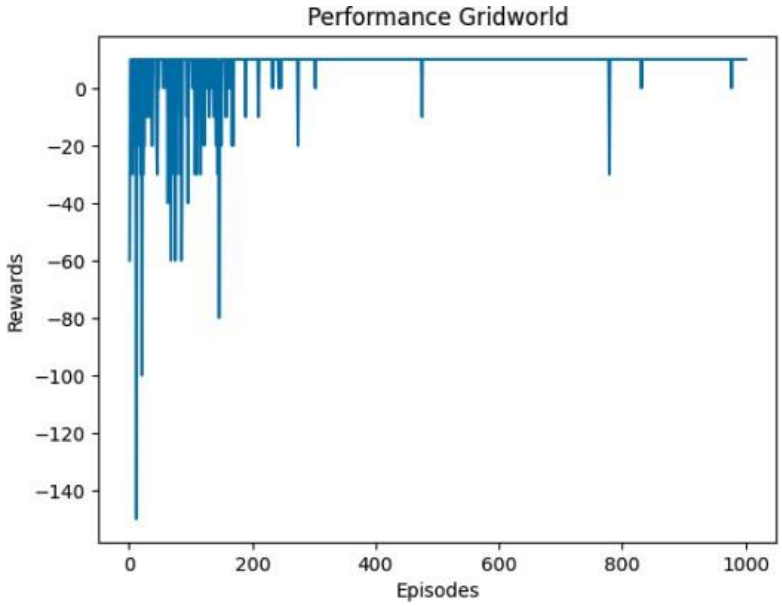

Figure 14: Reward for GridWorld alpha=0.001

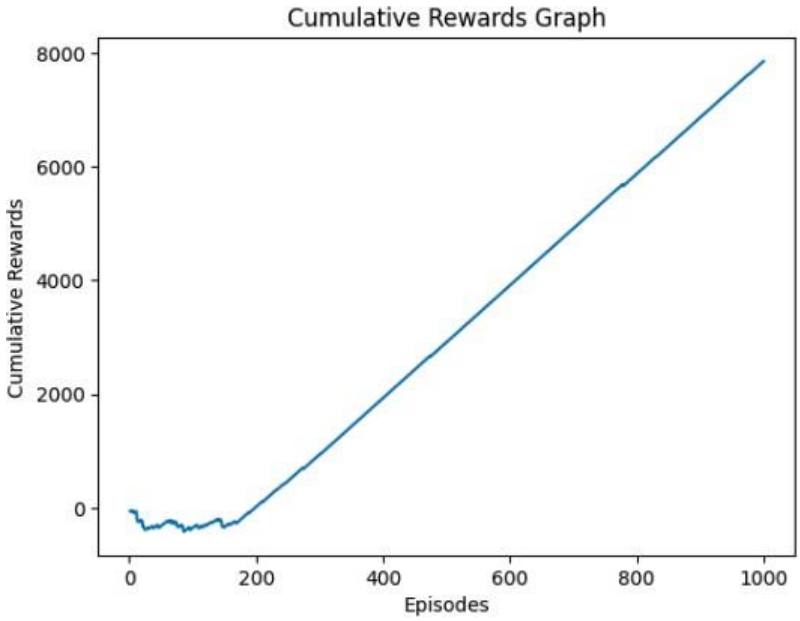

Figure 15: Cumulative Reward for GridWorld alpha=0.001

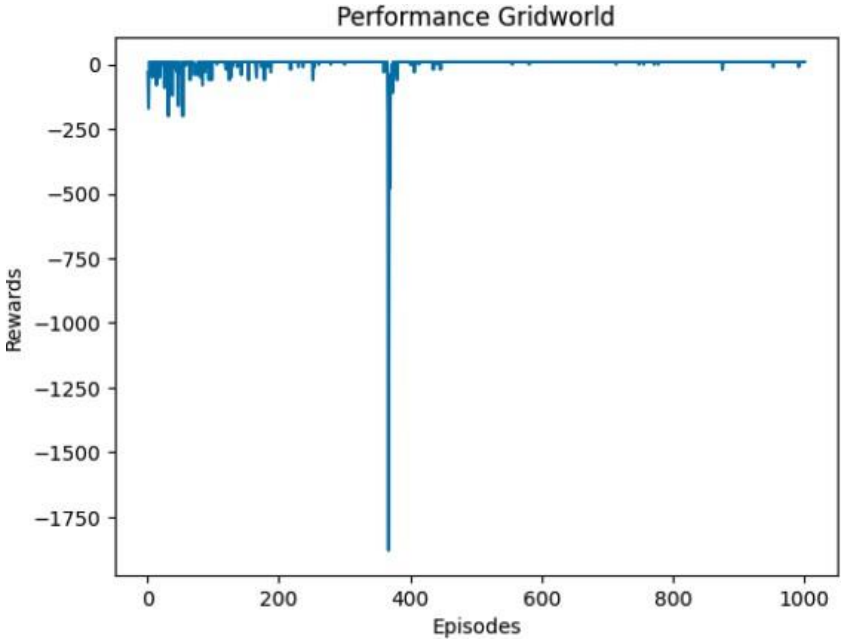

Figure 16: Reward for GridWorld alpha=0.01

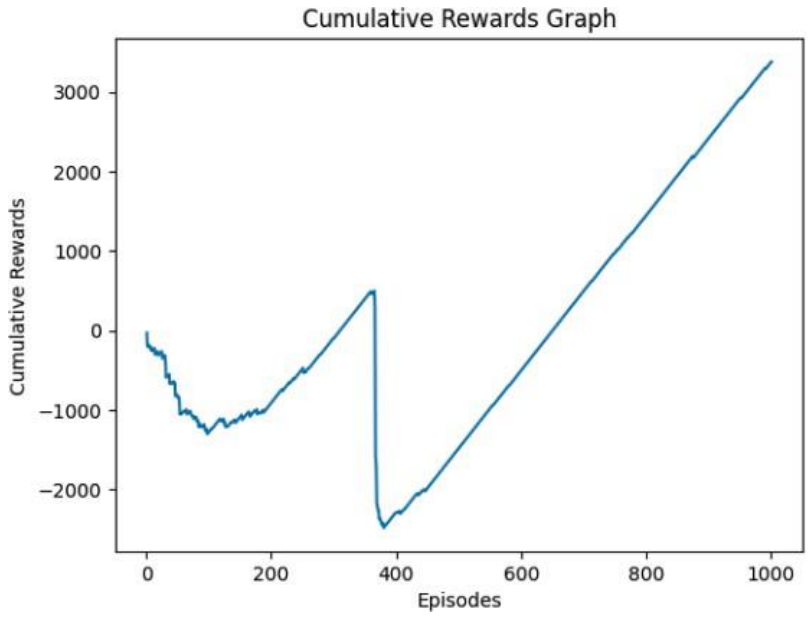

Figure 17: Cumulative Reward for GridWorld alpha=0.01

4 One-step Actor-Critic

Actor-Critic is a reinforcement learning algorithm which is a Temporal Difference(TD) version of Policy gradient. It has two networks: Actor and Critic. The actor takes the state as the input and decides which is the best action to take.The critic evaluates the action by computing the value function.It informs the actor how good was the action and how to adjust it.Both the learning of the actor and critic happen independent of each other.As time passes,the actor is producing better and better actions and the critic is getting better at evaluating the actions.
The weights are updated by computing the temporal difference(TD) between estimated return given a particular state and the value function computed by the value network at that particular state.The estimate of value function at the given state is evaluated using $v\hat{}(S, w)$, and for the next state is evaluated using $\hat{v}(S', w)$. The temporal difference is given as:
$\delta = R + v\hat{}(S, w) - \gamma * \hat{v}(S', w)$
The actor and critic networks are implemented using Fourier series feature representation for its the ease of gradient computation and effective way of modelling the states.
All the graphs are computed over 5 runs.

Pseudo Code:

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

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

Figure 18: One step Actor-critic algorithm

4.1 CartPole

1. Feature Representation: Used Fourier series for its ease of gradient computation and its ability to approximate any smooth value function.
2. Discount Parameter (λ): Used discount parameter $\lambda = 1$ as specified in the MDP.
3. σ : 1 and used softmax policy to pick an action.With $\sigma = 1$, there is a balance between both exploration and exploitation.The agent will explore different actions while still favoring actions with higher estimated values.
4. Step Size (α^{w}): The step size helps in updating the weights of the value function computed by the value network.Larger step size results in faster convergence but with higher number of oscillations.Smaller value takes longer to converge.I tried with alphas in the range [.001,.01,.8,8].The value of .01 gave the fastest convergence to the right value with least number of deviation in the end.
5. Step Size (α^{θ}): The step size helps in updating the weights of the policy function computed by the policy network.Larger step size results in faster convergence but with higher number of oscillations.Smaller value takes longer to converge.I tried with alphas in the range [.001,.01,.8,8].The value of .01 gave the fastest convergence to the right value with least number of oscillations in the end.
6. Fourier series order(k) : I experimented with different values for the order parameter, ranging from 1 to 10, specifically using the sequence [1, 3, 5, 10]. I noticed that higher order values effectively captured the complexities of the state, but there was a trade-off – as the order increased, the algorithm took significantly longer time to run. During 600 episodes, I observed that setting the order to 1 resulted in convergence, reaching a value of 500 in approximately 300 episodes.Values of 3 and 5 were much faster to converge at about 20 50 episodes but it took much longer for a single episode to run.I chose a value of 1 for the same reason.

The below curves are based on $\alpha^{w} = 0.01, \alpha^{\theta} = 0.01, \text{features}_{k} = 1, \text{episodes} = 600, \gamma = 1$.

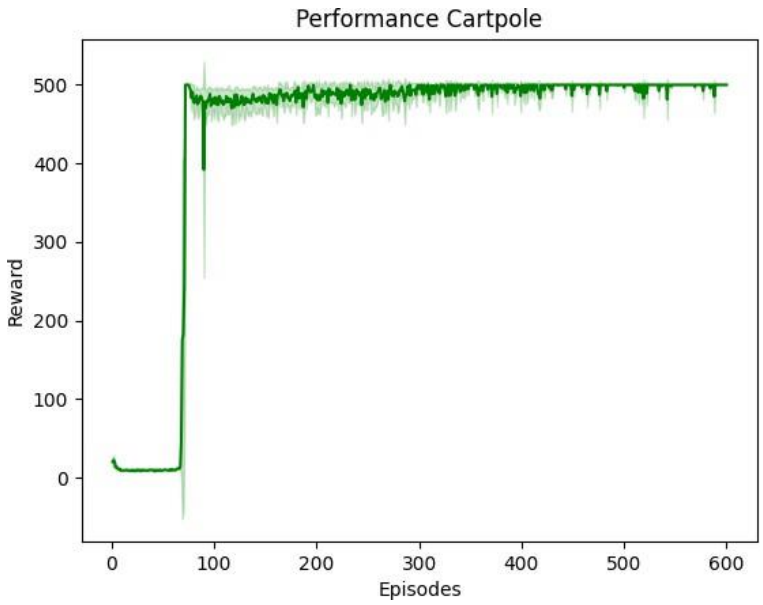

Figure 19: Reward for Cartpole

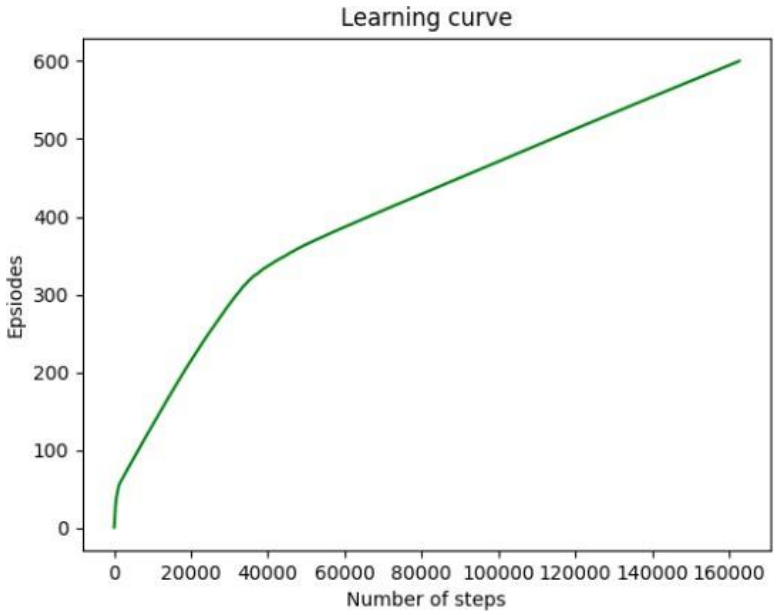

Figure 20: Learning curve for Cartpole

The curve below shows the value of the reward function with varying the values of alpha.

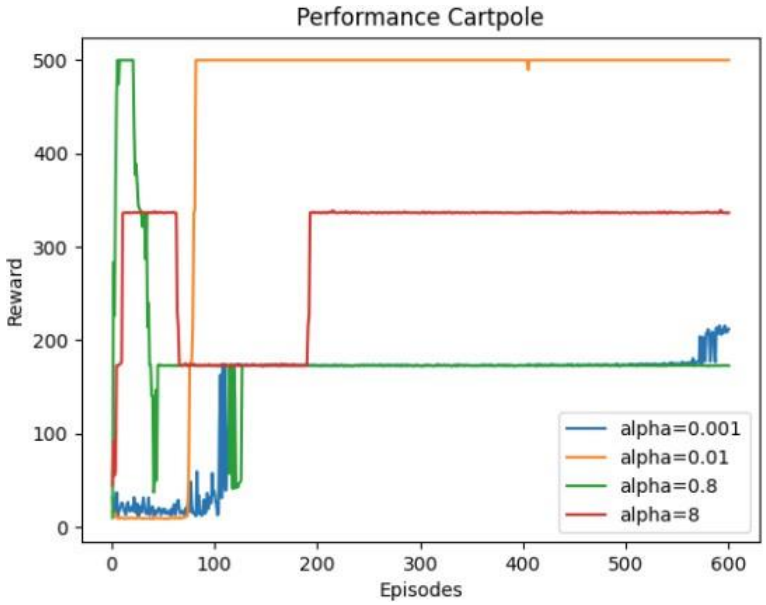

Figure 21: Reward for cartpole with varying alphas.For lower values of alpha, the curve doesn't converge.For higher values of alpha the behaviour is not definite.For alpha = .001 the curve shows optimal behaviour

The curve below shows the value of the reward with varying values of order.

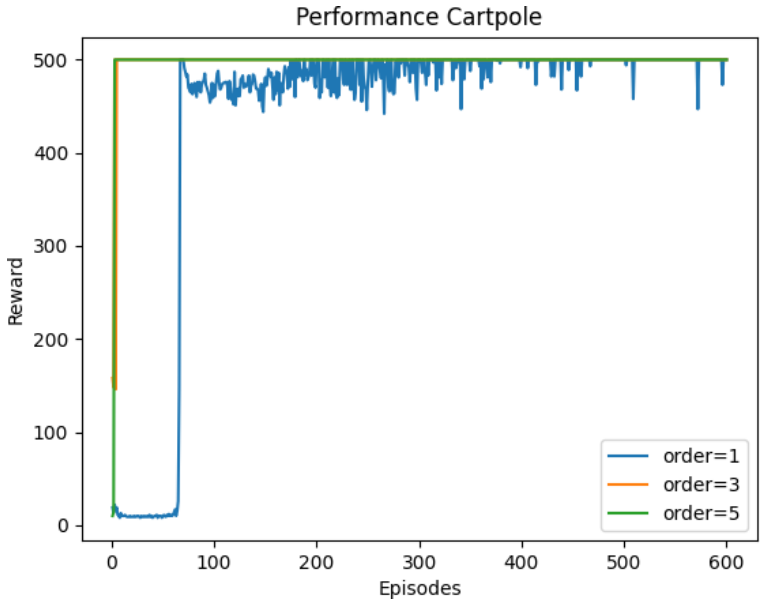

Figure 22: Reward for cartpole with varying feature order.For larger order values,the reward shoots up quickly but each episode takes longer.So,order 1 is chosen.

The curve below shows the value of the learning curve function with varying the values of feature order.

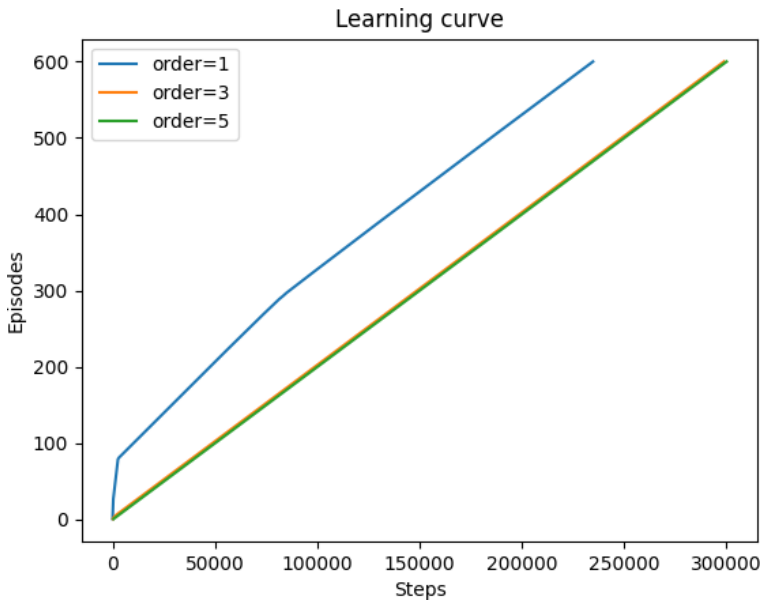

Figure 23: Learning for cartpole with varying feature order.With order=1,learning is the fastest.For higher orders,learning takes longer.

4.2 Acrobot

1. Feature Representation: Used Fourier series for its ease of gradient computation and its ability to approximate any smooth value function.
2. Discount Parameter (λ): Used discount parameter $\lambda = 1$ as specified in the MDP.
3. σ:1 and used softmax policy to pick an action.With $\sigma = 1$,there is a balance between both exploration and exploitation.The agent will explore different actions while still favoring actions with higher estimated values
4. Step Size (α^w): The step size helps in updating the weights of the value function computed by the value network.Larger step size results in faster convergence but with higher number of oscillations.Smaller value takes longer to converge.I tried with alphas in the range [.0001,.001,.01,.1,8].The value of .001 gave the fastest convergence to the right value with least number of deviation in the end.
5. Step Size (α^θ): The step size helps in updating the weights of the policy function computed by the policy network.Larger step size results in faster convergence but with higher number of oscillations.Smaller value takes longer to converge.I tried with alphas in the range [.0001,.001,.01,.1,8].The value of .001 gave the fastest convergence to the right value with least number of oscillations in the end.
6. Fourier series order(k) : I experimented with different values for the order parameter, specifically using the sequence [1, 3]. I noticed that higher order values effectively captured the complexities of the state, but there was a trade-off – as the order increased, the algorithm took significantly longer time to run. During 1000 episodes, I observed that setting the order to 1 resulted in convergence with slight deviation by 1000 episodes.Values of 3 took a lot of time since the number of terms in the feature vector is $(k + 1)^n$ which is 4^6.I therefore chose a value of 1.

The below curves are based on $\alpha^{\omega} = 0.001, \alpha^{\theta} = 0.001, \text{features}_k = 1, \text{episodes} = 1000, \gamma = 1$.

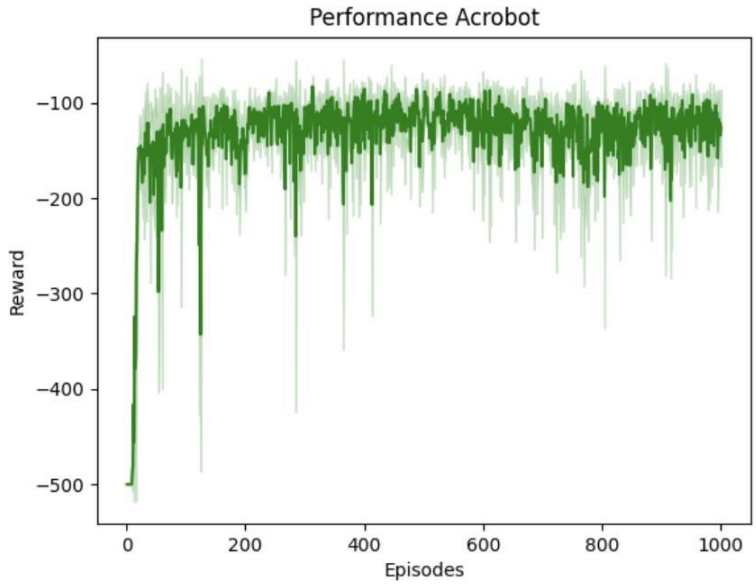

Figure 24: Reward for Acrobot

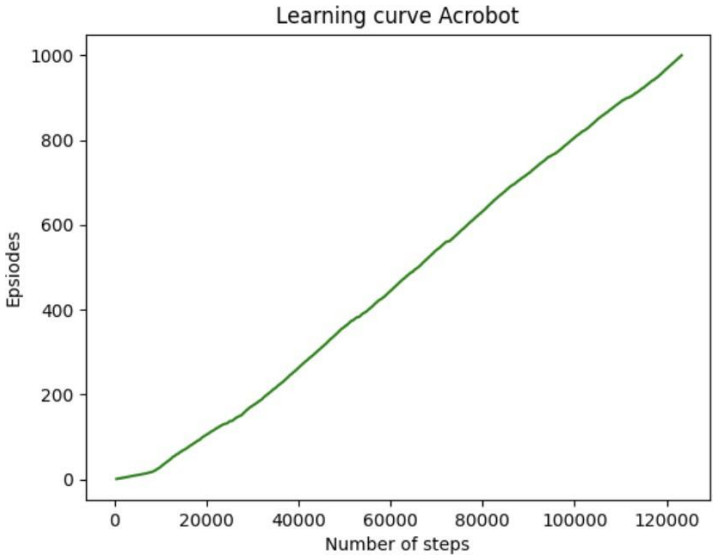

Figure 25: Learning curve for Acrobot

The curve below shows the value of the reward function with varying values of alpha.

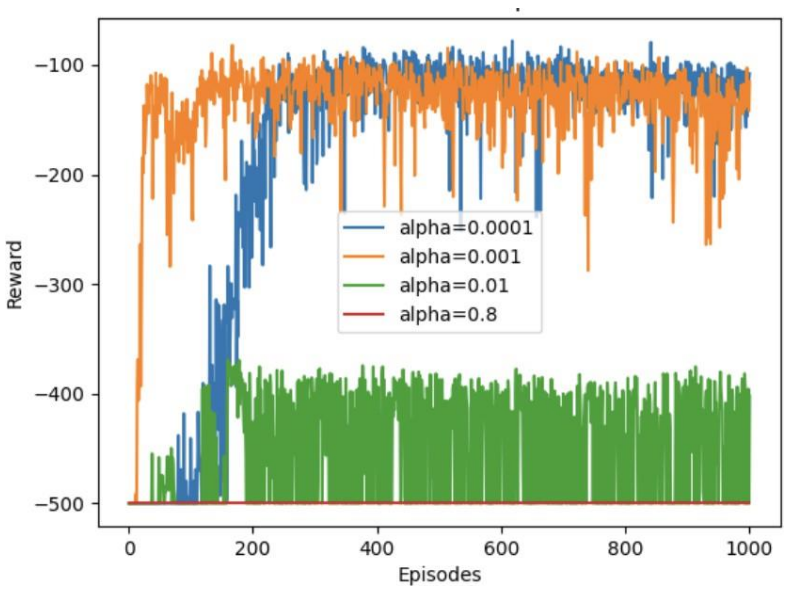

Figure 26: Reward for Acrobot with varying alphas.For higher values of alpha,the graph oscillates and doesn't converge to the optimal value.For very small values of alpha it takes longer to reach the optimal value compared to when alpha = .001

4.3 GridWorld

1. Feature Representation: Used Fourier series for its ease of gradient computation and its ability to approximate any smooth value function.
2. Discount Parameter (λ): Used discount parameter $\lambda = 0.9$ as specified in the MDP.
3. σ : 1 and used softmax policy to pick an action.With $\sigma = 1$, there is a balance between both exploration and exploitation.The agent will explore different actions while still favoring actions with higher estimated values
4. Step Size (α^{w}): The step size helps in updating the weights of the value function computed by the value network.Larger step size results in faster convergence but with higher number of oscillations.Smaller value takes longer to converge.I tried with alphas in the range [.0001,.001,.01,.8].The value of .001 gave the fastest convergence to the right value with least number of deviation in the end.
5. Step Size (α^{θ}): The step size helps in updating the weights of the policy function computed by the policy network.Larger step size results in faster convergence but with higher number of oscillations.Smaller value takes longer to converge.I tried with alphas in the range [.001,.01,.8,8].The value of .001 gave the fastest convergence to the right value with least number of oscillations in the end.
6. Fourier series order(k) :I chose a value of 1 for order since there are only a very few states.During 1000 episodes, I observed that setting the order to 1 resulted in convergence with very little deviation by 1000 episodes.

The below curves are based on $\alpha^{\omega} = 0.001, \alpha^{\theta} = 0.001, \text{features}_k = 1, \text{episodes} = 1000, \gamma = 0.9$.

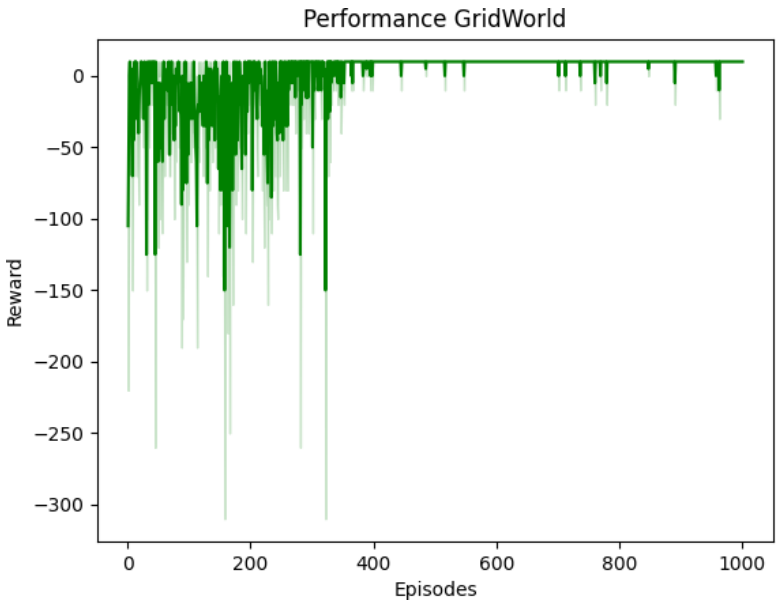

Figure 27: Reward for GridWorld

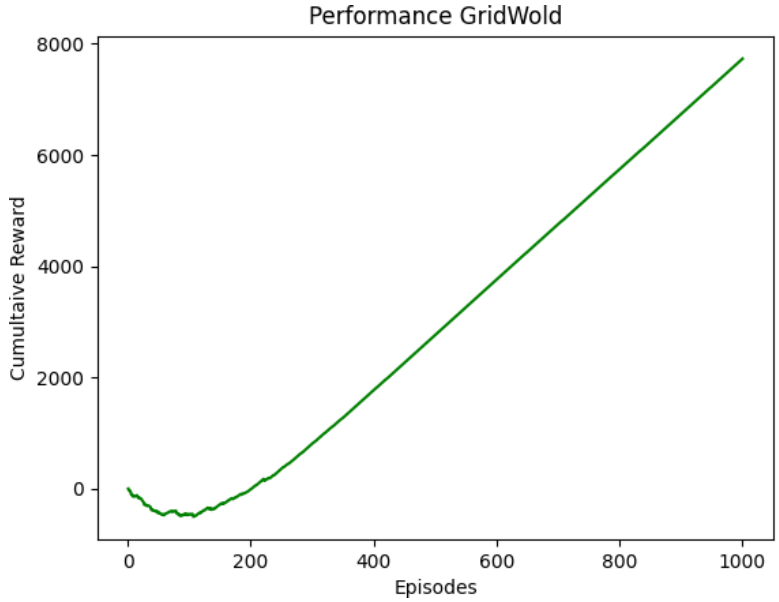

Figure 28: Cumulative reward for Grid World

The curve below shows the value of the reward function with varying values of alpha.

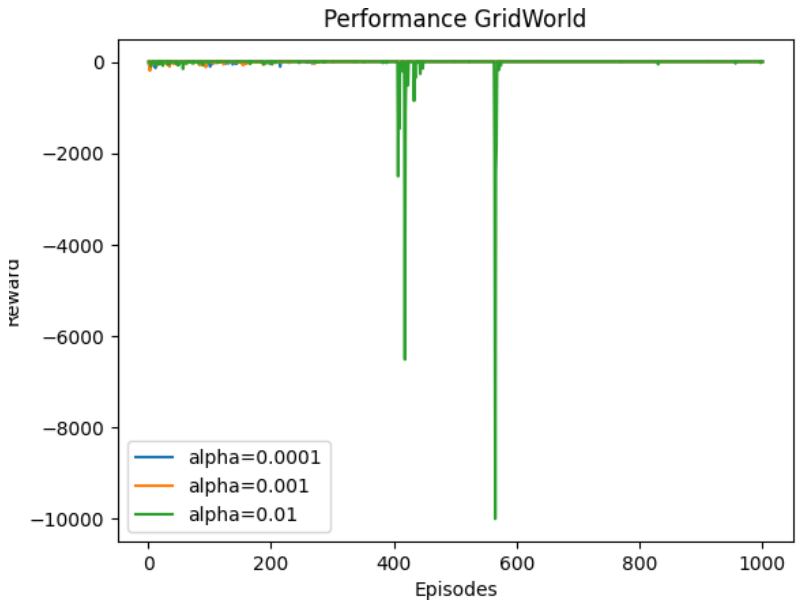

Figure 29: Reward for Grid world with varying alphas