REINFORCEMENT LEARNING ALGORITHMS: AN OVERVIEW

Avinash Amballa [*], **Gayathri Akkinapalli** [*]
Department of Computer Science
University of Massachusetts, Amherst
{aamballa,gakkinapalli}@umass.edu

ABSTRACT

This paper thoroughly investigates five Reinforcement Learning algorithms—Reinforce with baseline, One-step Actor-Critic, Semi-gradient SARSA, Semi-gradient N-step SARSA, and Tabular Dyna-Q. The exploration includes detailed implementation insights presented through pseudo-code, fine-tuning methodologies, and the resulting outcomes. These RL algorithms are tested on benchmark environments like CartPole, Acrobat, and 687-Grid-World. Code is available at [1].

1 ENVIRONMENT (MDP)

1.1 ACROBAT

Description: The setup comprises two links arranged in a linear chain, with one end securely anchored. The joint connecting the two links is actuated. The objective is to exert torques on the actuated joint to raise the open end of the chain to a specified height, beginning from the initial state where it hangs downward.

Action Space: It is Discrete and deterministic representing the torque applied on the actuated joint that connects both the links. The possible actions are -1, 0, 1.

State Space: It is Continuous with 6 values which provides the details about the two joint angles and their respective angular velocities. The state parameters being (θ_1, θ_2, ω_{θ_1}, ω_{θ_2}). The state representation variables are namely $\cos\theta_1$, $\sin\theta_1$, $\cos\theta_2$, $\sin\theta_2$, ω_{θ_1}, ω_{θ_2} whose range of values are (-1,1),(-1,1),(-1,1),(-1,1), (-12.57,12.57), (-28.27,28.27) respectively.

Rewards: The objective is to efficiently elevate the free end to a designated target height, minimizing the number of steps. Steps not leading to the goal receive a reward of -1, while reaching the target height terminates the process with a reward of 0. Default reward threshold is set at -100.

Start State: The parameters of the state representation (θ_1, θ_2, ω_{θ_1}, ω_{θ_2}) are uniformly chosen between the values -1 to 1 leaving both the links pointing downwards with little stochasticity in the initialization.

Termination: Episode terminates if any among the below occurs:
1. The free end reaches target height, that is by obeying the condition $-\cos\theta_1 - \cos(\theta_1 + \theta_2) \geq 1$
2. When Episode length reaches 500

Implementation: We use openai gym module to implement this environment.

1.2 CART POLE:

Description: A pole is attached to a cart by an un-actuated joint that moves along a frictionless track. The pendulum is initially positioned upright on the cart, and the objective is to maintain balance by exerting forces in the left and right directions on the cart.

[*]Equal contribution
[1]https://github.com/AmballaAvinash/RLalgorithms_summary

Action Space: It is discrete with 2 possible values 0,1, representing pushing the cart to left and right respectively.

State Space: It is continuous with 4 values for representing each state, that has the information about the position, velocity, pole angle and pole angular velocity. The range of the values are (-4.8, 4.8), $(-\infty, \infty)$, (-0.418 radians, 0.418 radians) and $(-\infty, \infty)$ respectively.

Reward: Given the objective of maintaining the pole upright for an extended duration, a reward of +1 is assigned for each step, encompassing the termination step. The reward threshold is 475.

Start State: All the values are uniformly chosen from (-0.05, 0.05).

Termination: Episode terminates if any among the below occurs:
1. Pole Angle is not in the range (-12°,12°)
2. Cart position is not in the range (-2.4, 2.4)
3. Episode length reaches 500

Implementation: We use openai gym module to implement this environment.

1.3 687 GRID-WORLD (DETERMINISTIC)

Description: This environment consists of a 5×5 grid world where each state $s = (r, c)$ describes the current coordinates/location of the agent. The grid world is surrounded by walls. If the agent hits a wall, it stays in its current state. There are two *Obstacle states* in this domain: one in state $(2, 2)$ and one in state $(3, 2)$. If the agent hits an obstacle, it stays in its current state. The agent cannot enter an Obstacle state. There is a *Water state* located at $(4, 2)$. There is a *Goal state* located at $(4, 4)$. All transition probabilities are deterministic.

Action Space: There are 4 possible actions: left, right, up, down

State Space: State space is the all possible states that an agent can enter.

Reward: The reward is always 0, except when transitioning to (entering) the Goal state, in which case the reward is 10; or when transitioning to (entering) the Water state, in which case the reward is -10.

Start State: The agent can start at a state other than obstacles.

Termination: Episode terminates if the agent reaches the goal state.

Implementation: We created this environment in python.

2 ALGORITHMS

The algorithms presented in this paper are sourced from the referenced book Sutton & Barto (2018)

2.1 REINFORCE WITH BASELINE

Description: The REINFORCE with Baseline algorithm (1) is an extension of basic REINFORCE which uses complete return and updates are made after the episode is completed. So it is a Monte Carlo algorithm and is only defined for episodic cases. The REINFORCE with Baseline algorithm incorporates a baseline, typically in the form of a state-value function or a learned critic network. This baseline helps reduce the variance in the gradient estimates, leading to more faster and stable learning.

2.2 ONE-STEP ACTOR-CRITIC (EPISODIC)

Description: The one-step actor–critic algorithm(2), in contrast to REINFORCE with baseline , utilizes the state-value function to estimate the value of both the current (first) and next(second) states in a transition. This allows for the computation of the one-step return (TD estimate), which is the discounted value of the second state added to the reward. While this algorithm introduces bias, the one-step return is advantageous due to lower variance and computational efficiency. This

Algorithm 1 Reinforce with baseline

Require: a differentiable policy parameterization $\pi(a|s,\theta)$
Require: a differentiable state value function parameterization $\hat{v}(s,w)$
Algorithm Parameters: step sizes $\alpha_\theta \geq 0, \alpha_w \geq 0$
Initialize Policy Parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

for each episode (Loop forever) **do**
Generate an episode S_0, A_0, R_1,....., S_{T-1}, A_{T-1}, R_T, following $\pi(.|.,\theta)$

for Loop for each step of the episode t=0,1,2,...,T-1: **do**
$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
$\delta \leftarrow G - \hat{v}(S_t, w)$
$w \leftarrow w + \alpha_w \delta \nabla \hat{v}(S_t, w)$ (**gradient update for value network**)
$\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta \nabla \log \pi(A_t|S_t,\theta)$ (**gradient update for policy network**)

end for
end for

approach, where the state-value function evaluates actions, is known as the actor–critic method, distinguishing the role of the state-value function as the critic and policy estimator as actor.

Algorithm 2 One step Actor Critic

Require: a differentiable policy parameterization $\pi(a|s,\theta)$
Require: a differentiable state-value function parameterization $\hat{v}(s,w)$
Parameters: step sizes $\alpha_\theta \geq 0, \alpha_w \geq 0$
Initialize Policy Parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$

for Loop forever for each episode **do**
Initialize S (first state of episode)
$I \leftarrow 1$
for each time step; while S is not terminal state **do**
$A \sim \pi(.|S,\theta)$
Take action A, Observe S',R
$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$
$w \leftarrow w + \alpha_w \delta \nabla \hat{v}(S, w)$ (**gradient update for value or critic network**)
$\theta \leftarrow \theta + \alpha_\theta I \delta \nabla \ln \pi(A|S,\theta)$ (**gradient update for policy or actor network**)
$I \leftarrow \gamma I$
$S \leftarrow S'$
end for
end for

2.3 Episodic Semi-Gradient SARSA

Description: This method is also called as Episodic semi-gradient one-step SARSA (3) , which does a general gradient-descent update for the action-value function. Here it does semi-gradient prediction to approximate action-value function, $\hat{q} \approx q_*$ which is in a parameterized functional form with weight vector w. The training examples are of the form $S_t, A_t \rightarrow U_t$ where the update target U_t is an approximation of $q_\pi(S_t, A_t)$, including one-step SARSA return. Here $U_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t)$ Pseudo-code for the complete algorithm is below.

2.4 Episodic Semi-Gradient n-step SARSA

Description: It is an n-step version of Episodic semi-gradient SARSA (4) by using an n-step return as the update target in the semi-gradient SARSA update equation. The n-step update equation is

Algorithm 3 Semi-Gradient SARSA

Require: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
  Algorithm Parameters: step sizes $\alpha \geq 0, \epsilon \geq 0$
  Initialize Value-function weights $w \in \mathbb{R}^d$ arbitrarily
  for each episode **do**
    $S, A \leftarrow$ initial state and action of episode (e.g., $\epsilon - greedy$)
    for each step of episode **do**
      Take Action A, observe R, S'
      if S' is Terminal **then**
        $w \leftarrow w + \alpha[R - \hat{q}(S, A, w)]\nabla\hat{q}(S, A, w)$
        Go to Next Episode
      end if
      Choose A' as a function of $\hat{q}(S', ., w)$ (e.g., $\epsilon - Greedy$)
      $w \leftarrow w + \alpha[R + \gamma\hat{q}(S', A', w) - \hat{q}(S, A, w)]\nabla\hat{q}(S, A, w)$
      $S \leftarrow S'$
      $A \leftarrow A'$
    end for
  end for

$w_{t+n} = w_{t+n-1} + \alpha[G_{t:t+n} - \hat{q}(S_t, A_t, w_{t+n-1})]\nabla\hat{q}(S_t, A_t, w_{t+n-1})$ where $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1}R_{t+n} + \gamma^n\hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1})$ The pseudocode for the same is below.

Algorithm 4 Semi-Gradient n-step SARSA

Require: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
  Algorithm Parameters: step sizes $\alpha \geq 0, \epsilon \geq 0$ a positive integer n
  Initialize Value-function weights $w \in \mathbb{R}^d$ arbitrarily
  All Store and access operations (S_t, A_t and R_t) can take their index mod n+1
  for each episode **do**
    Initialize and store $S_0 \neq terminal$
    Select and store an action $A_0 \sim \epsilon - Greedy$ w.r.t $\hat{q}(S_0, .w)$
    $T \leftarrow \infty$
    for t = 0,1,2,... **do**
      if $t \leq T$ **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 $A_{t+1} \sim \epsilon - Greedy$ w.r.t $\hat{q}(S_{t+1}, ., w)$
        end if
      end if
      $\tau \leftarrow t - n + 1$ (τ **is the time whose estimate is being updated**)
      if $\tau \geq 0$ **then**
        $G \leftarrow \sum_{i=\tau+1}^{min(\tau+n,T)} \gamma^{i-\tau-1}R_i$
        if $\tau + n < T$ **then**
          $G \leftarrow G + \gamma^n\hat{q}(S_{\tau+n}, A_{\tau+n}, w)$
        end if
        $w \leftarrow w + \alpha[G - \hat{q}(S_\tau, A_\tau, w)]\nabla\hat{q}(S_\tau, A_\tau, w)$
      end if
      Until $\tau = T - 1$
    end for
  end for

2.5 TABULAR DYNA-Q

Description: This algorithm (5) merges model-free Q-learning with model-based planning to efficiently learn an optimal policy in a Markov Decision Process (MDP). While model-free learning updates Q-values based on real experiences, Dyna-Q also maintains a learned model of the environment, enabling it to simulate transitions and rewards. In Dyna-Q, the acting, model-learning, and direct RL processes require little more computation time. Dyna-Q implements Q-learning for n=0 case. See the below pseudo code for complete implementation of the algorithm.

Algorithm 5 Tabular Dyna-Q

Require: Initialize Q(s,a) and Model(s,a) for all $s \in S$ and $a \in A(s)$

for Loop forever **do**
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \epsilon - Greedy(S, Q)$
  (c) Take action A; observe resultant reward, R and state, S'
  (d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
  (e) $Model(S, A) \leftarrow R, S'$ (Assuming deterministic environment)
  for n times **do**
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
  end for
end for

3 Results and Discussion

3.1 Cart Pole

We implemented Reinforce with baseline, One-step Actor Critic, Semi Gradient SARSA, Semi Gradient n-step SARSA algorithms. We used Adam optimizer with number of trails = 5, episodes = 2000, $\gamma = 0.99$. We reported the total mean reward across those 5 trails for every episode (see below graphs for results). For neural networks, we took a single hidden layer neural network (with 128 neurons) for both policy (last layer softmax) and value network where we initialized the weights randomly. For hyper-parameter tuning we use random search and grid search.

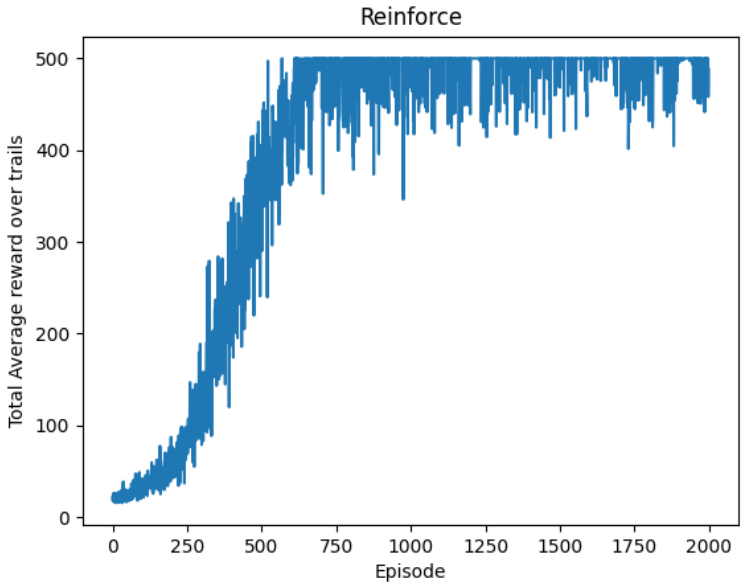

Figure 1: Reinforce with Baseline on Cartpole. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$

Figures 1 and 2 show the performance of Reinforce algorithm with baseline on Cartpole environment whose α_θ and α_w values are (0.0001,0.0001) and (0.0001,0.001) respectively. Clearly, we can see the this algorithm achieved an optimal performance (reward = 500), and also we can see that the baseline is successful in reducing the variance (consistent). We experimented with $\alpha_\theta = \{0.01, 0.001, 0.0001\}, \alpha_w = \{0.01, 0.001, 0.0001\}$ and we observed a better performance and convergence when $\alpha_\theta \leq \alpha_w$. Other combination of α_θ and α_w were either having poor learning, or having lot of variance with minimal consistency.

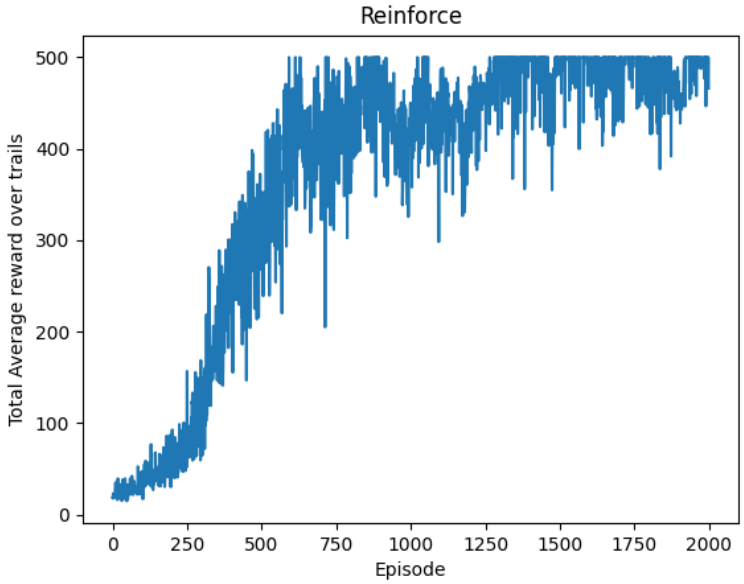

Figure 2: Reinforce with Baseline on Cartpole. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.001$

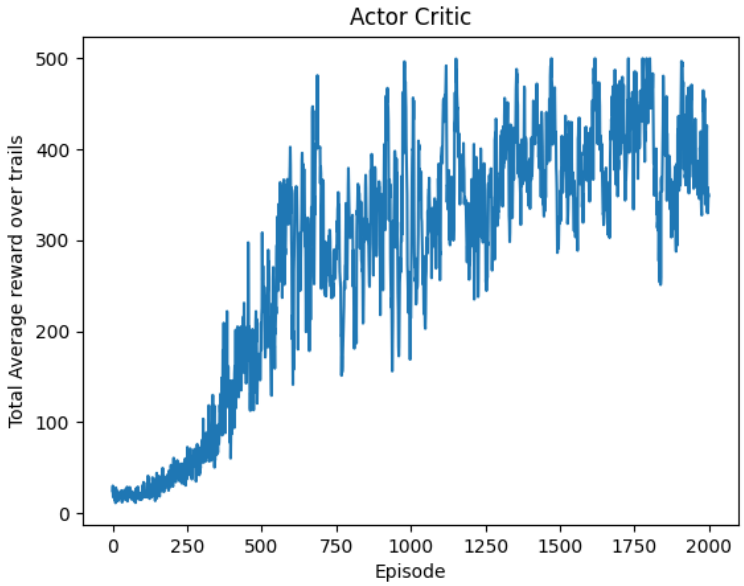

Figure 3: one step Actor Critic on Cartpole. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.001$

Figure 3 represent the performance of One-step actor critic algorithm on Cartpole environment. From the figure, we can tell that this algorithm achieved an near-optimal performance (rewards$\geq$400). We experimented with $\alpha_\theta = \{0.01, 0.001, 0.0001\}, \alpha_w = \{0.01, 0.001, 0.0001\}$ and we observed a better performance and convergence when $\alpha_\theta < \alpha_w$. Other combination of α_θ and α_w were either having poor learning, or having lot of variance with minimal consistency and sometimes when the consistency is achieved the maximum reward obtained is less than 500 (not an optimal nor near-optimal policy).

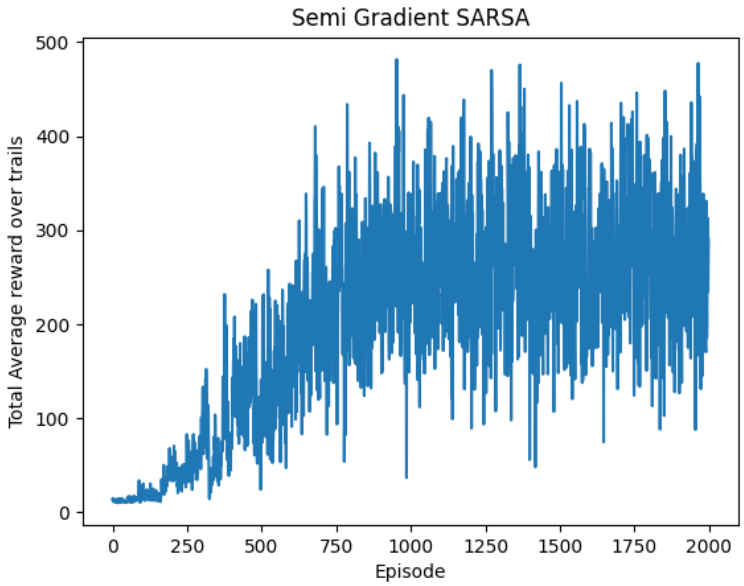

Figure 4: Semi Gradient SARSA on Cartpole. Here $\alpha = 0.001$, $\epsilon = 0.4$

Figure 4 represent the performance of Semi Gradient SARSA on Cartpole environment. From the graph we can observe that the algorithm achieved a near-optimal performance (reward > 300). We experimented with $\alpha = \{0.01, 0.001, 0.0001\}, \epsilon = \{0.1, 0.2, 0.3, 0.4\}$. The pattern we observed while conducting the hyper parameter tuning is that, lower value of ϵ like $\{0.1,0.2,0.3\}$ were producing less consistent output, while increasing the ϵ was effecting the learning of the algorithm (reduced learning). On the other hand, α value of 0.01 was reaching the consistency very fastly with the maximum reward obtained less than 500 (reward 500 was obtained nearly none to very rare number of times), while α of 0.0001 was not learning well, with a random pattern of learning curve. So the ideal combination we could obtain for α and ϵ is (0.001,0.4).

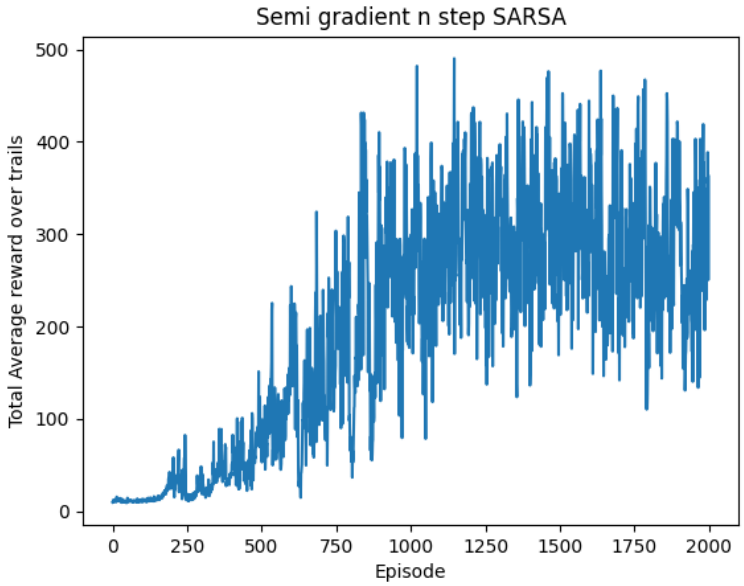

Figure 5: Semi Gradient n-step SARSA on Cartpole. Here $\alpha = 0.001$, $\epsilon = 0.3, n = 5$

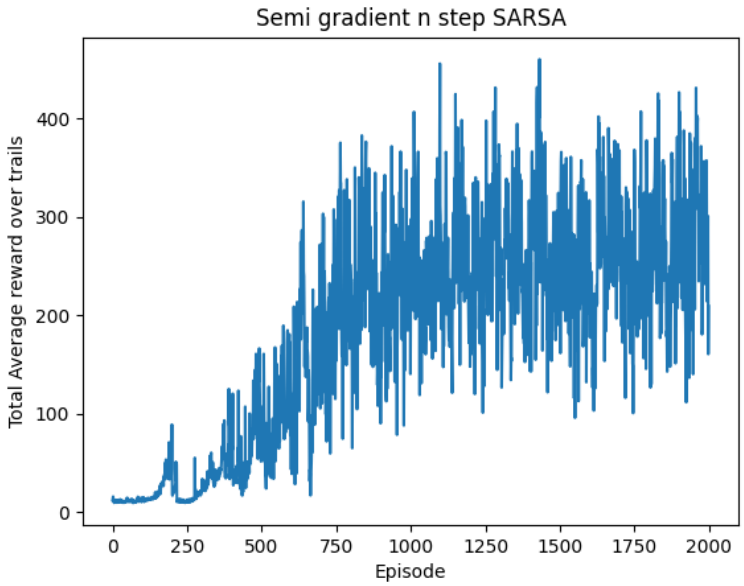

Figure 6: Semi Gradient n-step SARSA on Cartpole. Here $\alpha = 0.001$, $\epsilon = 0.3, n = 10$

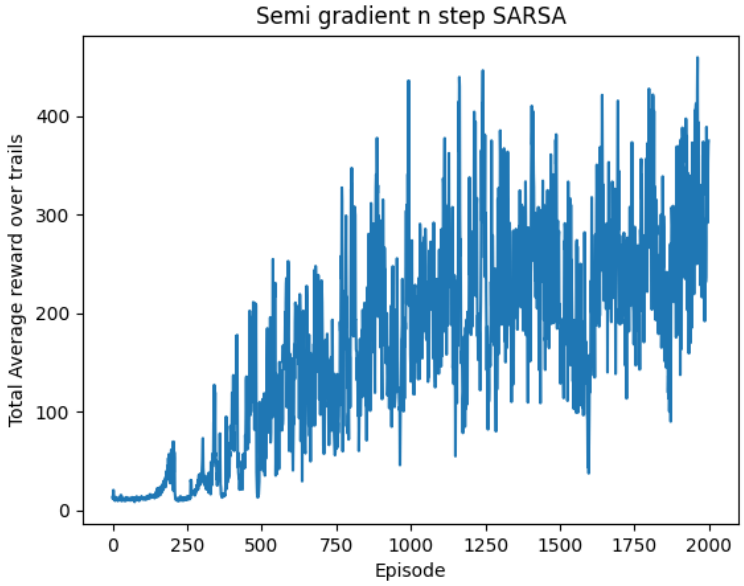

Figure 7: Semi Gradient n-step SARSA on Cartpole. Here $\alpha = 0.001$, $\epsilon = 0.3, n = 15$

Figures 5, 6, 7 shows the the learning curves for Semi-Gradient n-step SARSA for n=5,10,15 respectively. From the figure, we can tell that this algorithm achieved an near-optimal performance (rewards>300). We experimented with $\alpha = \{0.01, 0.001, 0.0001\}, \epsilon = \{0.3, 0.4\}, n = \{5, 10, 15\}$. As we increase "n," the algorithm takes a longer sequence of actions into account, potentially leading to a more accurate estimation of the value function. However, there is a trade-off, Larger values of "n" have resulted in slower updates (more time taking) and increased computational complexity, as the algorithm needs to consider a more extended sequence of experiences. During the process of hyperparameter tuning we found that having higher value of ϵ was reducing the learning while

keeping it very short is reducing the consistency. On the other hand, α value of 0.01 was coverging very quickly but not with the highest rewards but whereas the value of 0.0001 was having minimal to low learning. So the ideal combination we could obtain was (0.001,0.3) for α and ϵ.

3.2 ACROBAT

We implemented Reinforce with baseline, One-step Actor Critic, Semi Gradient SARSA, Semi Gradient n-step SARSA algorithms. We used Adam optimizer with number of trails = 5, episodes = 2000, $\gamma = 0.99$. We reported the total mean reward across those 5 trails for every episode (see below graphs for results). For neural networks, we took a single hidden layer neural network (with 128 neurons) for both policy (last layer softmax) and value network where we initialized the weights randomly. For hyper-parameter tuning we use random search and grid search.

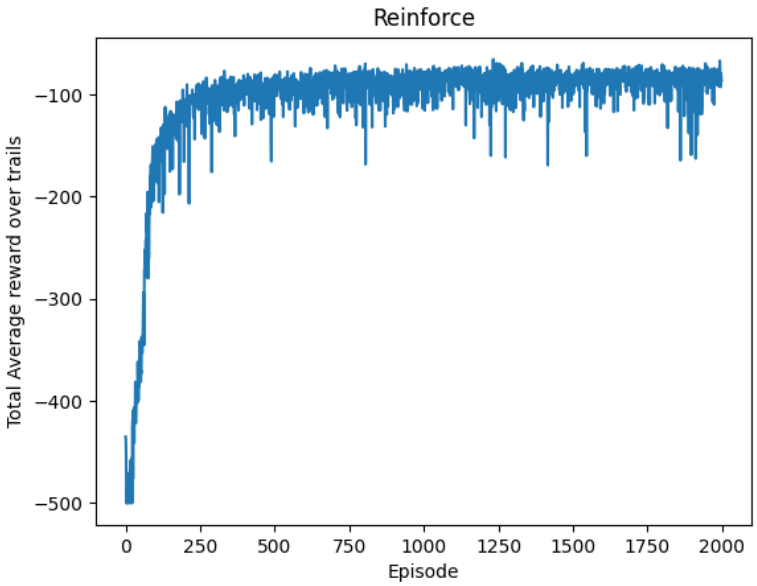

Figure 8: Reinforce with Baseline on Acrobat. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$

Figures 8 and 9 show the performance of Reinforce algorithm with baseline on Acrobat environment. Clearly, we can see the this algorithm achieved an optimal performance (reward $\geq$ -100), and baseline is also successful in reducing the variance. We experimented with $\alpha_\theta = \{0.01, 0.001, 0.0001\}, \alpha_w = \{0.01, 0.001, 0.0001\}$ and we observed a better performance and convergence when $\alpha_\theta \leq \alpha_w$. During the exploration, the values of α_θ and α_w being greater than or equal to 0.001 were facing the exploding gradient problem with poor learning. And by the above condition observed, the optimal hyper parameters we found were (0.0001,0.0001) and (0.0001,0.001) for α_θ and α_w respectively.

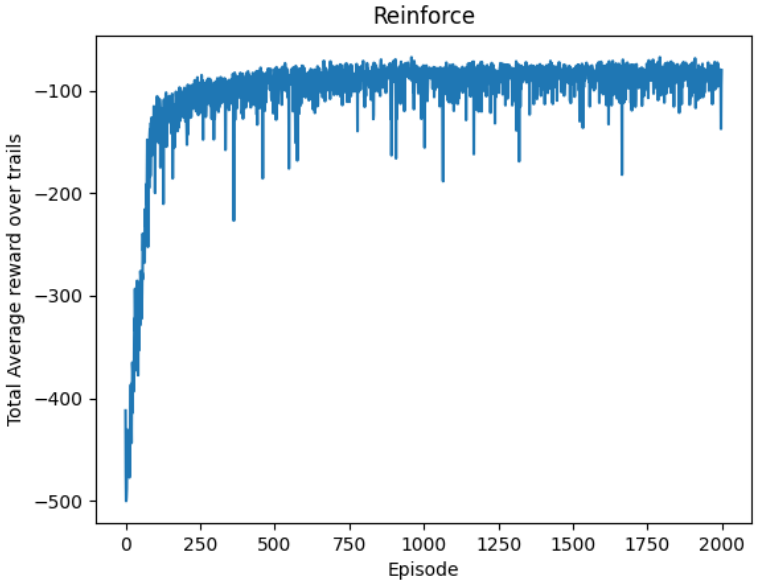

Figure 9: Reinforce with Baseline on Acrobat. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.001$

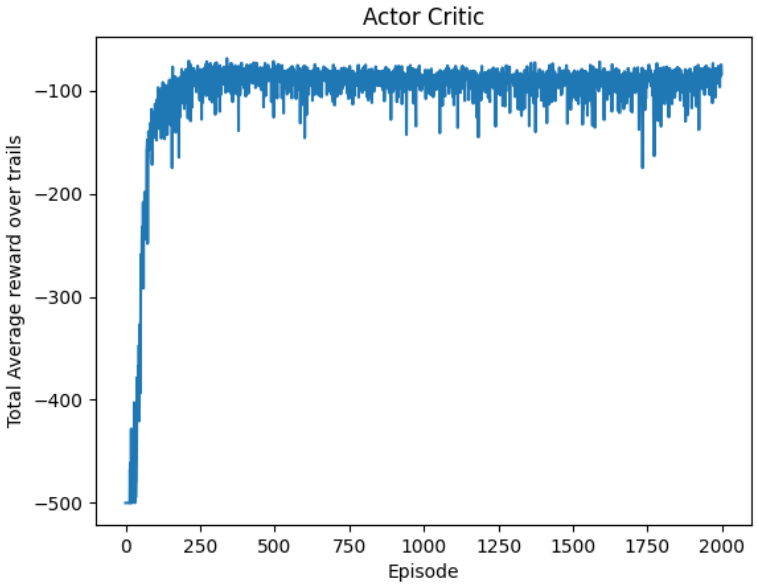

Figure 10: one step Actor Critic on Acrobat. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.0001$

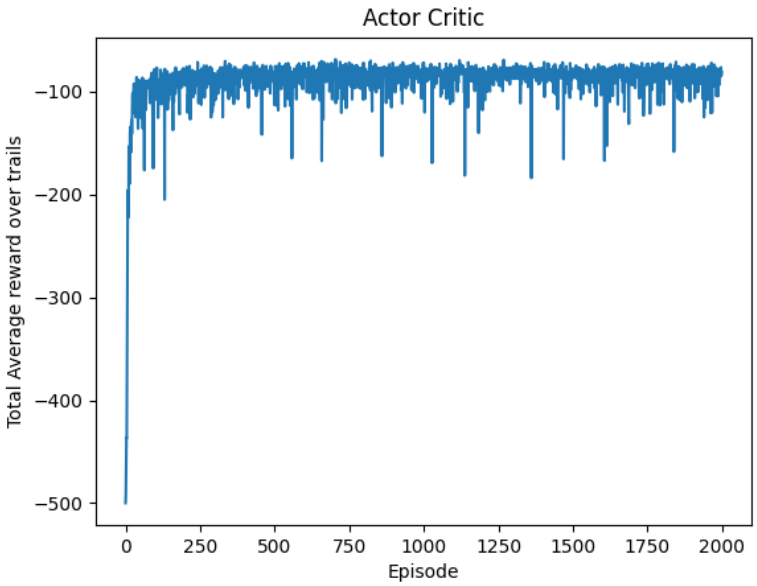

Figure 11: one step Actor Critic on Acrobat. Here $\alpha_\theta = 0.0001$, $\alpha_w = 0.001$

Figure 10, 11 represent the performance of One-step actor critic algorithm on Acrobat environment. From the figure, we can tell that this algorithm achieved an optimal performance (rewards≥-100) . We experimented with $\alpha_\theta = \{0.01, 0.001, 0.0001\}, \alpha_w = \{0.01, 0.001, 0.0001\}$ and we observed a better performance and convergence when $\alpha_\theta \leq \alpha_w$. The pattern of $\alpha_\theta \leq \alpha_w$ we observed during our exploration for the optimal results for the Actor critic algorithm were similar to that of Reinforce, so the optimal hyperparameters we obtained were (0.0001,0.0001) and (0.0001,0.001) for α_θ and α_w respectively.

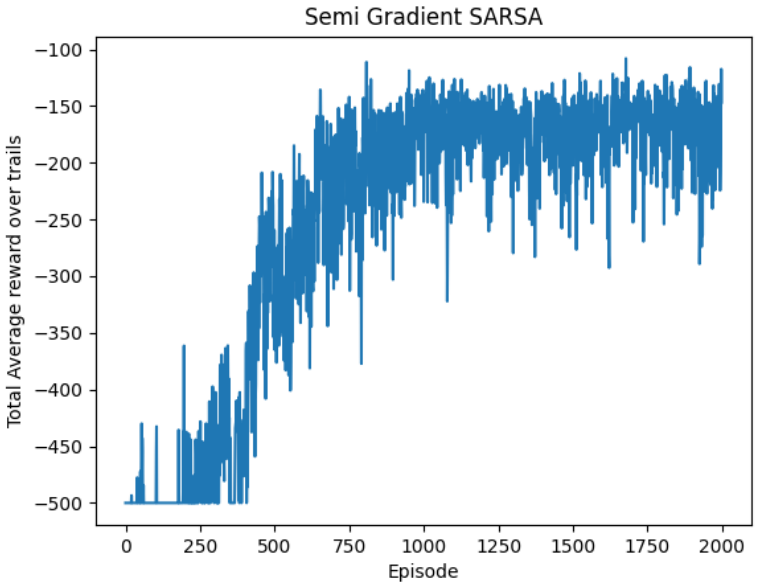

Figure 12: Semi Gradient SARSA on Acrobat. Here $\alpha = 0.01$, $\epsilon = 0.4$

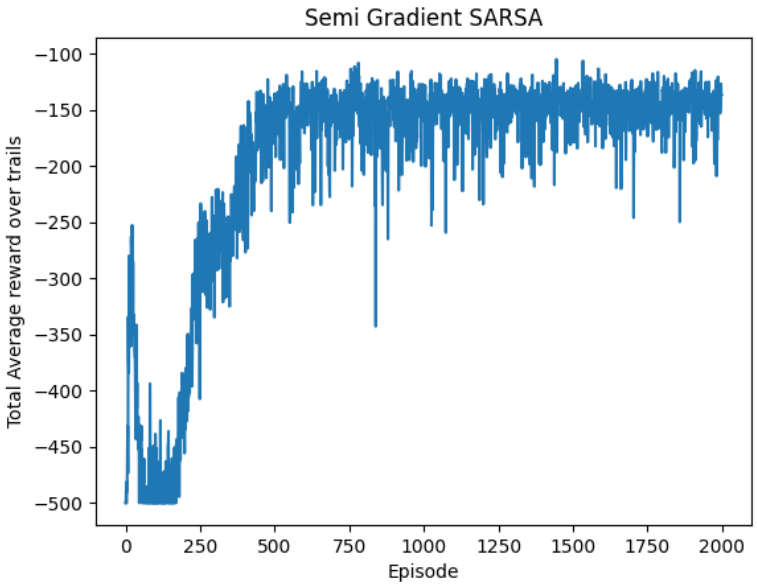

Figure 13: Semi Gradient SARSA on Acrobat. Here $\alpha = 0.001$, $\epsilon = 0.4$

Figure 12,13 represent the performance of Semi Gradient SARSA on Acrobat environment. From the graph we can observe that the algorithm achieved a near-optimal performance (reward > -150). We experimented with $\alpha = \{0.01, 0.001, 0.0001\}, \epsilon = \{0.1, 0.2, 0.3, 0.4\}$ During our exploration we found that by increasing the ϵ value was having least consistency outputs while reducing the same were effecting the learning since we are reducing the exploration factor. While the α value decides the learning rate, increasing it's value was converging fastly with suboptimal rewards and reducing the value were having minimal to no learning pattern in the output. So the ideal combination of the hyperparameters we found were (0.01,0.4) and (0.001,0.4) of α, ϵ respectively.

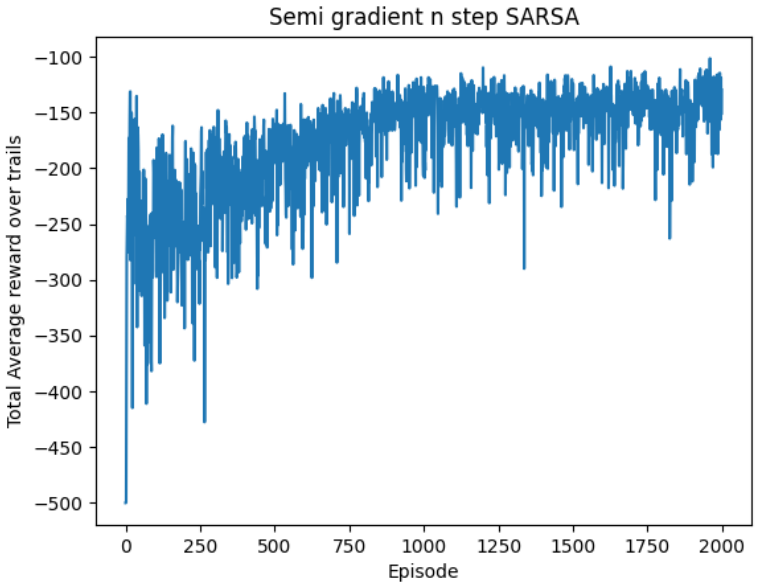

Figure 14: Semi Gradient n-step SARSA on Acrobat. Here $\alpha = 0.001$, $\epsilon = 0.3, n = 5$

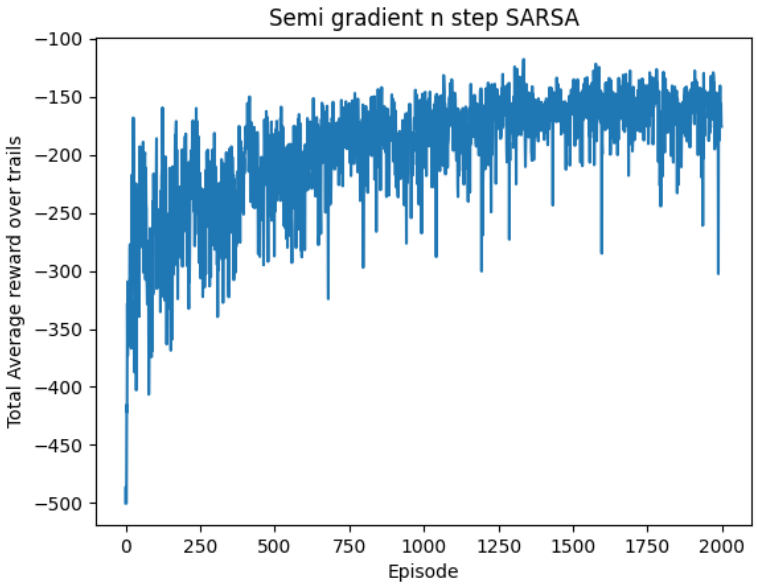

Figure 15: Semi Gradient n-step SARSA on Acrobat. Here $\alpha = 0.001$, $\epsilon = 0.4, n = 5$

Figures 14, 15 shows the the learning curves for Semi-Gradient n-step SARSA. From the figure, we can tell that this algorithms achieved an near-optimal performance (rewards>-150). We experimented with $\alpha = \{0.01, 0.001, 0.0001\}, \epsilon = \{0.3, 0.4\}, n = \{5\}$. The insights we obtained on the patterns of α and ϵ were almost similar as the Semi gradient SARSA.

3.3 687-GRID WORLD

We implement Dyna-Q algorithm on 687-Grid world deterministic environment. In addtion, we run value iteration on this environment to obtain the optimal policy and compute the MSE between the optimal policy and the learned policy by Dyna-Q. For hyper-parameter tuning we use random search most of time (explored Grid search as well).

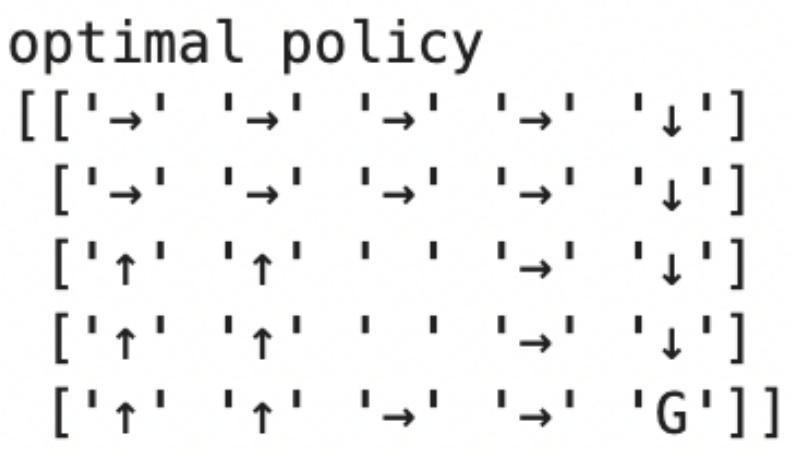

Figure 16: Greedy policy learned by Dyna-Q algorithm on 687-Gridworld. Here $\alpha = 0.3$, $\epsilon = 0.6, n = 5, T = 10000, \gamma = 0.9$

Figure 16 represent the greedy policy learned by Dyna-Q algorithm. We can see that this is an optimal policy indeed. We experimented with $\alpha = \{0.2, 0.3, 0.4\}$, $\epsilon = \{0.3, 0.4, 0.5, 0.6\}, n = \{5, 10\}$

```
optimal value function
[[ 4.783   5.3144  5.9049  6.561   7.29  ]
 [ 5.3144  5.9049  6.561   7.29    8.1   ]
 [ 4.783   5.3144  0.      8.1     9.    ]
 [ 4.3047  4.783   0.      9.     10.    ]
 [ 3.8742  4.3047  9.     10.      0.    ]]
optimal policy
[['→' '→' '→' '→' '↓']
 ['→' '→' '→' '→' '↓']
 ['↑' '↑' ' ' '→' '↓']
 ['↑' '↑' ' ' '→' '↓']
 ['↑' '↑' '→' '→' 'G']]
Number of iterations 11
```

Figure 17: Optimal value function, optimal policy and number of iterations for value iteration algorithm on 687-grid world

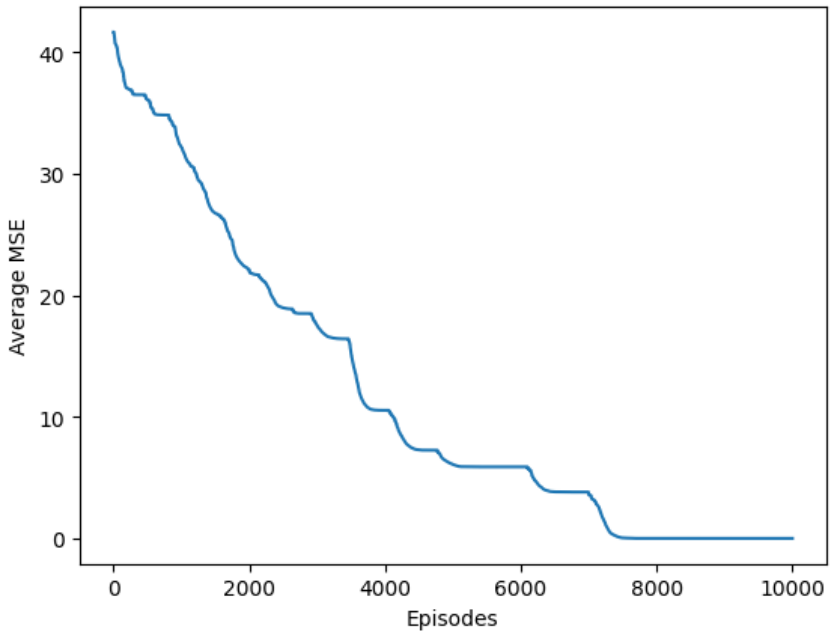

Figure 18: Average MSE of the learned value function (by DynaQ) and optimal value function over 20 trails Vs Episodes

Figure 18 shows the Average MSE Vs epsiodes plot for Dyna-Q algorithms. We can see that this loss gradually decreases and reached zero around 7000 episodes, this indicated that the optimal value function has been achieved and indeed optimal policy has been achieved.

REFERENCES

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.