Reinforcement Learning Project Report

Shruti Ramesh Babu Mallavolu, Susmita Madineni

Introduction

In our project, we have implemented three different Reinforcement Learning algorithms - Semi-gradient n-step SARSA, True Online SARSA, Prioritized Sweeping. We implemented Semi-gradient n-step SARSA, True Online SARSA algorithms on three different MDPs - Mountain Car Domain, CartPole domain and Acrobot domain and implemented Prioritized Sweeping on Grid World.

1.Episodic Semi-Gradient n-step SARSA

Algorithm Description

The Episodic Semi-Gradient n-step SARSA uses a parameterized policy to approximate the q value. This algorithm is more suitable for environments with continuous states. This means that the state space could have infinite states, which makes it extremely difficult to tabulate the state-actions. Hence, they are approximated as parameterized action-value functions - $\hat{q}(s, a, \mathbf{w})$, $w \in \mathbb{R}$ and it can be calculated by -
$\hat{q}(s, a, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a) = \sum_{i=1}^{d} w_i x_i(s, a)$, for each pair of state s and action a.

The return is calculated in a bootstrapping way and is given by -
$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1})$, where $t + n \leq T$

The n-step update equation is given by -
$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})]\nabla \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})$, where $0 \leqq t \leq T$

Since, the q approximation appears in both the target and the prediction, the method is semi-gradient. Here, we used Fourier basis to update the weights in order to approximate q.

The policy is chosen in an ϵ - greedy way depending on the current value function. So, it selects the action with maximum returns most of the time, but a random action with ϵ - probability.
Basically, the algorithm first samples an action using ϵ - greedy policy and calculates a reward until the terminal state. Next, it updates the weights based on the difference between the n-step discounted reward in the episode and the action-value function estimate which is also called bootstrapping. Thus, it tends to perform better with a value of $n > 1$.

Pseudocode

The pseudocode for Episodic Semi-Gradient n-step SARSA is given in Figure 1 (taken from Sutton and Barto book):

1.1 Mountain Car Environment

Hyperparameters Tuning

The hyperparameters tuning for Mountain Car are described as below -

- α - Step Size
  We have used a smaller step size as this has helped the algorithm to learn better. A higher step size increased the weights by a large magnitude which in-turn decreased the learning and performance of the agent. A decaying step size did not increase the performance of the algorithm, and hence we used a fixed size of α

- ϵ - Exlporation Rate
  The ϵ set to a lower value is returning better results. This could be because with a higher value of ϵ, the learning is lower and exploring is higher. Hence, it could pick random actions with a higher probability that may not

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: Semi-gradient n-step SARSA

necessarily be good for increasing the performance of the agent. It may also result in the agent trying for more number os steps before terminating, which could also increase the time taken.

- Order of Fourier basis
  We have set the order of Fourier Basis to be a low value as the learning takes a long time for a higher order. Especially having to compute around 2000 episodes with such a high order requires more computational power. Hence, we used a lower order of Fourier basis.
- Weights Initialization
  We have initialized the weights to zero.

Experimental Results

Based on the analysis above, I got the best performance on substituting the values below. There is a high variance involved because it is taking the algorithm a very higher number of episodes and a higher order of Fourier basis to learn and converge to the best value. But, after 200 episodes we do notice the algorithm converging and hence upon more training the algorithm will perform much better.

- $\alpha = 0.04$
- $\epsilon = 0.1$
- Order of Fourier basis = 1
- $n = 4$

1.2 Acrobot Environment

Hyperparameters Tuning

The hyperparameters tuning for Acrobot are described as below -

- α - Step Size
  We have set a fixed α size to a very low value so that the weights do not explode on getting updated. So, it will become difficult for the algorithm to learn properly.
- ϵ - Exlporation Rate
  It is better to set the ϵ value to a lower value, because a higher exploration rate would mean that the algorithm explores more than exploiting and hence the discounted returns would be unpredictable and most often not optimal. Hence, it is better to have a lower value of ϵ

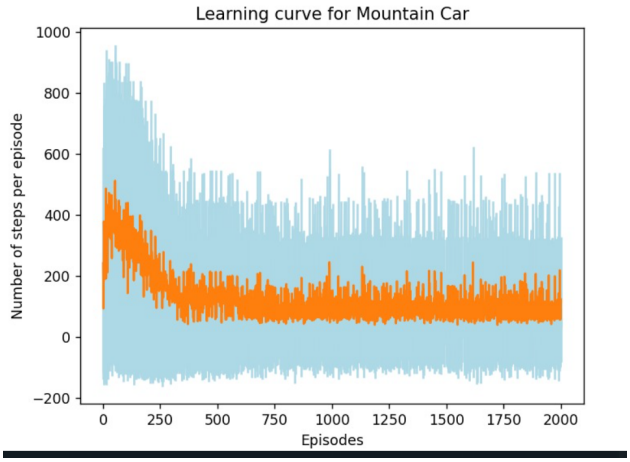

Figure 2

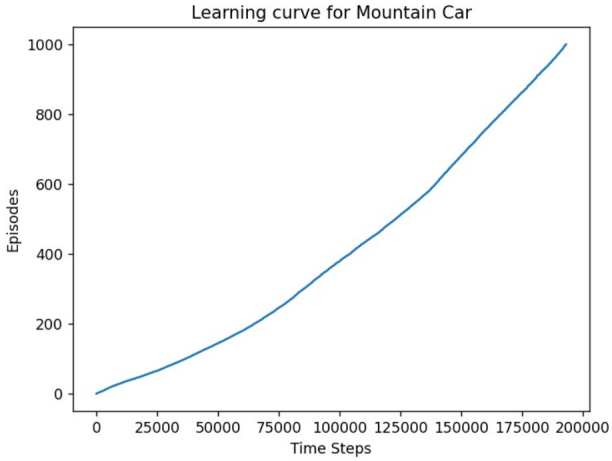

Figure 3

- Order of Fourier basis
  After trying many orders between 1 to 10, we have found that the algorithm learns faster with a lower order. As the order increases, the algorithm takes much longer to learn as the number of features in Acrobot is 6. Since it has so many features, it forms a $1x64$ matrix which would again increase the computation time.
- Weights Initialization
  We have chosen to initialize the weights to zero.

Experimental Results

Based on the analysis above, I got the best performance on substituting the values below. The variance is high because Acrobot has 6 features in its state space. Hence, computing a higher order of Fourier basis would essentially mean more computation time and power. But, we see the graph converging after some episodes and the performance would improve more with higher number of iterations.

- $\alpha = 0.001$
- $\epsilon = 0.1$
- Order of Fourier basis $= 1$
- $n = 4$

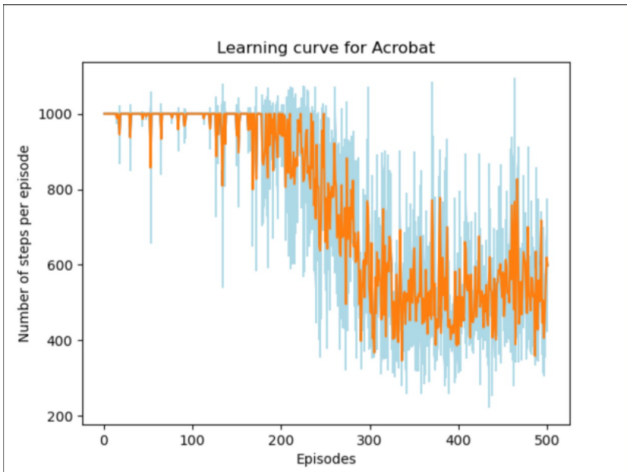

Figure 4

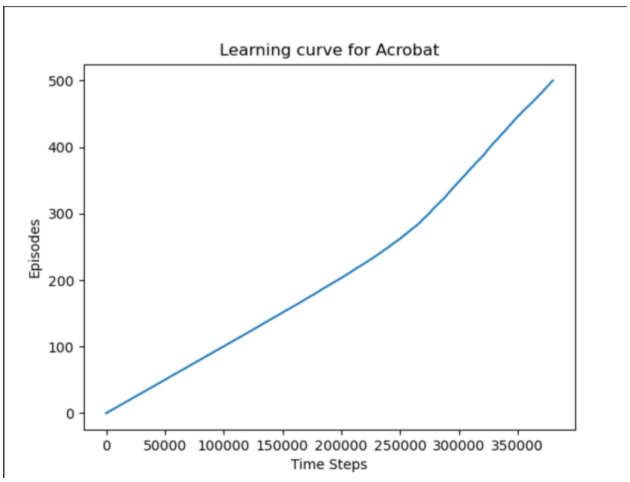

Figure 5

1.3 CartPole Environment

Hyperparameters Tuning

The hyperparameters tuning for CartPole are described as below -

- α - Step Size
  The step size is low because CartPole is very sensitive to step size. Hence, we found it better to have a lower value of fixed size α. A lower value ensures that the weights do not increase by a huge magnitude and hence the learning will be better.

- ϵ - Exlporation Rate
  We have taken a low value of ϵ as the discounted returns are much higher with a lower value of ϵ. This could be because the exploitation is more and the algorithm picks the best action more often. Hence, the discounted returns are much higher.

- Order of Fourier basis
  We had to initialize a higher order for CartPole as it seems to be converging for higher rewards than when the order is lower.

- Weights Initialization
  Initialized the weights to zero because the algorithm seems to be performing the same with random initialization of weights.

Experimental Results

Based on the analysis above, I got the best performance on substituting the values below. CartPole has 4 features in its state space, and taking a higher order would again mean that the computation would take longer to observe the convergence.

- $\alpha = 0.04$
- $\epsilon = 0.1$
- Order of Fourier basis = 5
- $n = 4$

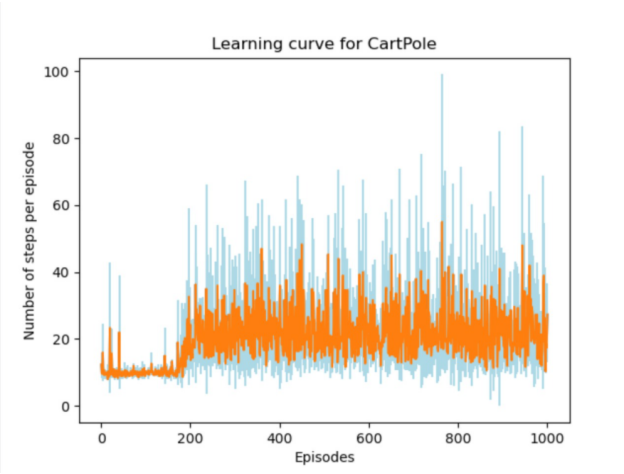

Figure 6

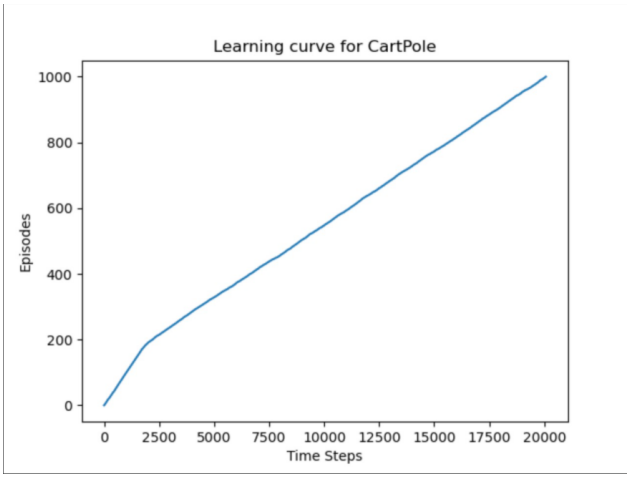

Figure 7

2.True Online SARSA

Algorithm Description

In True Online SARSA algorithm, we maintain an eligibility trace for each state, where we record the history of which states and actions the agent has visited and interacted. Consider an MDP where the agent travels from state $S1$ to $S2$, the eligibility trace will keep track of the states and actions it visited and performed in order to reach $S2$. It is an

improvement from SARSA n-step and Temporal Difference algorithms. This is helpful in scenarios where past experiences of the agent are used to make long-term decisions. The algorithm updates the action-value function after every action performed by the agent. With this, the agent can learn more efficiently by learning in every step and having an accurate representation of expected future rewards.

In this algorithm, eligibility traces is maintained for each state-action pair visited by the agent. At every time step, eligibility trace updation of current state-action pair takes place and for all the other state-action pairs, the eligibility traces are decayed(by multiplying lambda(decay factor) with eligibility traces). The term eligibility trace vector is computed using:

$e = \gamma\lambda e + (1 - \alpha\lambda\gamma(e^T\phi))\phi$ - λ is the trace decay rate where $\lambda \in [0, 1]$

Pseudocode

The pseudocode for True Online SARSA for estimating $w^T x$ or q_* is as follows(taken from Sutton and Barto book- Fig8):

True online Sarsa(λ) for estimating $\mathbf{w}^\top\mathbf{x} \approx q_\pi$ or q_*

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$, small $\varepsilon > 0$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose $A' \sim \pi(\cdot|S')$ or ε-greedy according to $\hat{q}(S', \cdot, \mathbf{w})$
    $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
    $Q \leftarrow \mathbf{w}^\top\mathbf{x}$
    $Q' \leftarrow \mathbf{w}^\top\mathbf{x}'$
    $\delta \leftarrow R + \gamma Q' - Q$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
    $Q_{old} \leftarrow Q'$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $A \leftarrow A'$
  until S' is terminal

Figure 8: TRUE Online SARSA

2.1 Mountain Car Environment

Hyperparameters Tuning

We have used linear function approximations like Fourier basis and employed ϵ-greedy policy exploration to choose the action with respect to action-value function while implementing the algorithm. The following hyperparameters needs to be tuned as part of the algorithm:

- Order of the Fourier basis - Order of Fourier basis determines the state representation's (ϕ) complexity. With higher N values, the state representation will become more complex and big since it encodes all the state information. N should be carefully selected since the size of the state representation will be exponentially increased. We experimented with different orders from 1 to 10 and observed that for few state feature parameters, with lesser order size, the agent learns and converges quickly. For higher order value, we observed that the agent is taking atleast 200 episodes to learn something for a low step size value to avoid exploring weights problem.

- Initialization of the weights - I have initialized the weights to zero. Any values greater than zero are increasing the runtime of the algorithm. We have computed state action values using weights and state features. With ϵ-greedy

policy, action with maximum q-values are taken, this will help the algorithm to learn quickly.

- Exploration parameter is ϵ-greedy - After looking into different strategies, I have got good results when I used a fixed small epsilon value with no decay. With every update of the weight vectors, the agent learns which action to pick and learns accordingly. By setting ϵ value to lower value using the greedy exploration, the algorithm converges.

- Step size(α) - Step size values determines the learning rate of w. A lower value of step size will lead to converge slowly while a higher value if step size leads to faster convergence of w. We can employ decaying step size after certain number of episodes and observe the learning, but I have not observed much difference when step size is decayed, so I have used a constant value.
- Trace decay rate (λ) - I have observed that using previous steps knowledge and having a slower decay rate(λ) value helped the agent in learning, so I chose $\lambda = 0.96$(close to 1).

Experimental Results

Based on the above analysis, the below parameters gave best results out of all my experiments:

- Order of the Fourier basis = 10
  We have tested multiple N values and observed that the algorithm is performing best for N = 10. As the value is increased, the performance increased and the runtime also increased significantly.

- Initialization of the weights = initialized to zero
- Exploration parameter is ϵ-greedy = 0.01
- Step size(α) = 0.0009
- Trace decay rate (λ) = 0.96

The following learning curve is obtained for the above set of hyperparameters:

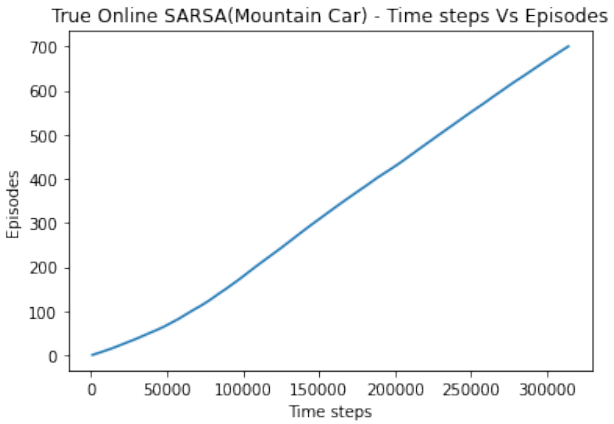

Figure 9: True Online SARSA(Mountain Car) - Time steps Vs Episodes

2.2 Acrobot Environment

Hyperparameters Tuning

We have used linear function approximations like Fourier basis and employed ϵ-greedy policy exploration to choose the action with respect to action-value function while implementing the algorithm. The following hyperparameters needs to be tuned as part of the algorithm:

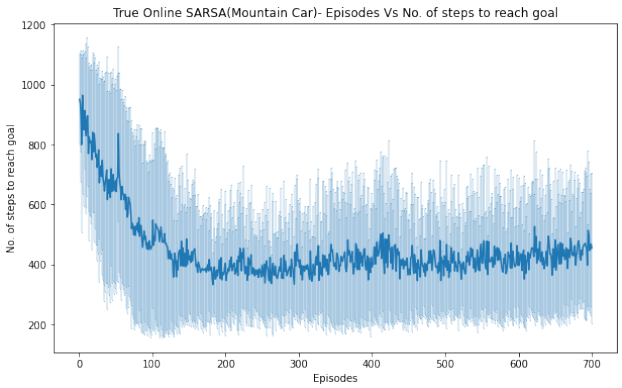

Figure 10: True Online SARSA(Mountain Car) - Episodes Vs No. of steps to reach goal

- Order of the Fourier basis - Order of Fourier basis determines the state representation's (ϕ) complexity. With higher N values, the state representation will become more complex and big since it encodes all the state information. N should be carefully selected since the size of the state representation will be exponentially increased. We experimented with different orders from 1 to 10 and observed that for few state feature parameters, with lesser order size, the agent learns and converges quickly. In case of Acrobot MDP, the state dimensions are 6(high), if the order value is too high, it takes more time to converge and finetuning the hyperparameters will also consume more time. So, I have fixed my order to 2.

- Initialization of the weights - I have initialized the weights to zero. Any values greater than zero are increasing the runtime of the algorithm. We have computed state action values using weights and state features. With ϵ-greedy policy, action with maximum q-values are taken, this will help the algorithm to learn quickly.

- Exploration parameter is ϵ-greedy - After looking into different strategies, I have got good results when I used a fixed small epsilon value with no decay. With every update of the weight vectors, the agent learns which action to pick and learns accordingly. By setting ϵ value to lower value using the greedy exploration, the algorithm converges.

- Step size(α) - Step size values determines the learning rate of w. In this MDP, I have observed that for higher step size values, the algorithm is not learning much since the weights are exploding. I found that with lower step size value, the algorithm is learning. In case of Acrobot MDP, the dimensions of the state increases exponentially if order is high and to avoid the weights exploding issue, I have used a small order size 2.
- Trace decay rate (λ) - I have observed that using previous steps knowledge and having a slower decay rate(λ) value helped the agent in learning, so I chose $\lambda = 0.95$(close to 1).

Experimental Results

Based on the above analysis, the below parameters gave best results out of all my experiments:

- Order of the Fourier basis = 2 We have tested multiple N values and observed that the algorithm is performing best for N = 2. As the value is increased, the performance increased and the runtime also increased significantly.

- Initialization of the weights = initialized to zero
- Exploration parameter is ϵ-greedy = 0.01
- Step size(α) = $1e^{-5}$
- Trace decay rate (λ) = 0.95

The following learning curve is obtained for the above set of hyperparameters:

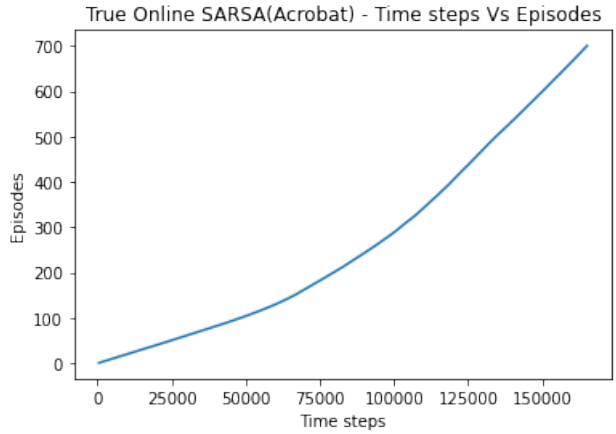

Figure 11: True Online SARSA(Acrobot) - Time steps Vs Episodes

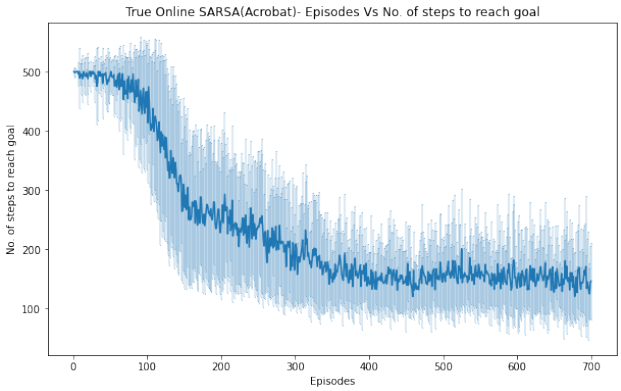

Figure 12: True Online SARSA(Acrobot) - Episodes Vs No. of steps to reach goal

2.3 Cart Pole Environment

Hyperparameters Tuning

We have used linear function approximations like Fourier basis and employed ϵ-greedy policy exploration to choose the action with respect to action-value function while implementing the algorithm. The following hyperparameters needs to be tuned as part of the algorithm:

- Order of the Fourier basis - Order of Fourier basis determines the state representation's (ϕ) complexity. With higher N values, the state representation will become more complex and big since it encodes all the state information. N should be carefully selected since the size of the state representation will be exponentially increased. We experimented with different orders from 1 to 10 and selected order =2 as the algorithm performed well.

- Initialization of the weights - I have initialized the weights to zero. Any values greater than zero are increasing the runtime of the algorithm. We have computed state action values using weights and state features. With ϵ-greedy policy, action with maximum q-values are taken, this will help the algorithm to learn quickly.

- Step size(α) - Step size values determines the learning rate of w. I have observed that choosing the Step size is very sensitive. In this MDP, I have observed that for higher step size values, the algorithm is not learning much since the weights are exploding. I found that with lower step size value, the algorithm is learning. In case of CartPole MDP, the dimensions of the state increases exponentially if order is high and to avoid the weights exploding issue, I have used a small order size.

- Trace decay rate (λ) - I have observed that using previous steps knowledge and having a slower decay rate(λ) value helped the agent in learning, so I chose $\lambda = 0.90$.

Experimental Results

Based on the above analysis, the below parameters gave best results out of all my experiments:

- Order of the Fourier basis = 2
- Initialization of the weights = initialized to zero
- Exploration parameter is ϵ-greedy = 0.01
- Step size(α) = $1e^{-5}$
- Trace decay rate (λ) = 0.90

The following learning curve is obtained for the above set of hyperparameters:

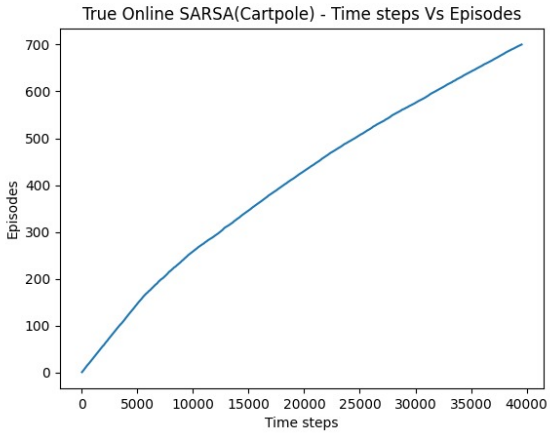

Figure 13: True Online SARSA(CartPole) - Time steps Vs Episodes

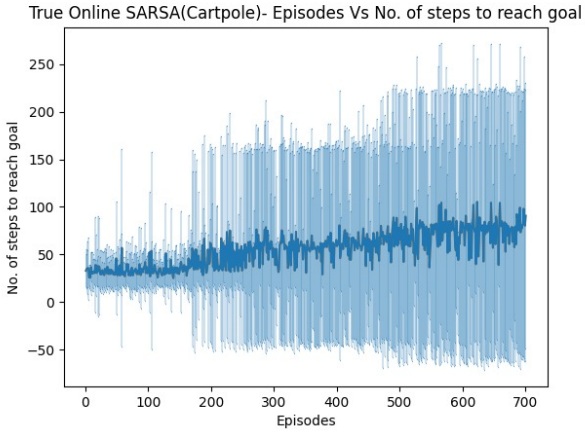

Figure 14: True Online SARSA(CartPole) - Episodes Vs No. of steps to reach goal

3.Prioritized Sweeping - EXTRA CREDIT

Algorithm Description

We have implemented the Prioritized Sweeping algorithm for a modified version of Gridworld. The Gridworld here is deterministic and the actions are found in the optimal way.

The algorithm basically forms a model of states and actions, and updates the q value based on the priority levels. Each model is given a priority and the state-action with the highest priority is selected.
The algorithm runs until the priority queue is empty.

Pseudocode

The pseudocode for Prioritized Sweeping is as follows(taken from Sutton and Barto book- Fig8):

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S,Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Model(S,A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
(f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
(g) Loop repeat n times, while $PQueue$ is not empty:
$S,A \leftarrow first(PQueue)$
$R,S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
Loop for all $\bar{S},\bar{A}$ predicted to lead to S:
$\bar{R} \leftarrow$ predicted reward for $\bar{S},\bar{A},S$
$P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.
if $P > \theta$ then insert $\bar{S},\bar{A}$ into $PQueue$ with priority P

Figure 15: Prioritized Sweeping

3.1 GRID World

Hyperparameters

The hyperparameter to consider here is the gridsize. The gridsize chosen here is 5. But, the performance keeps getting better for higher values of gridsize.

Experimental Results

The graph of Episodes vs Number of Steps is given below. The learning curve should be almost linear and the same performance should be showed when the gridsize is increased.

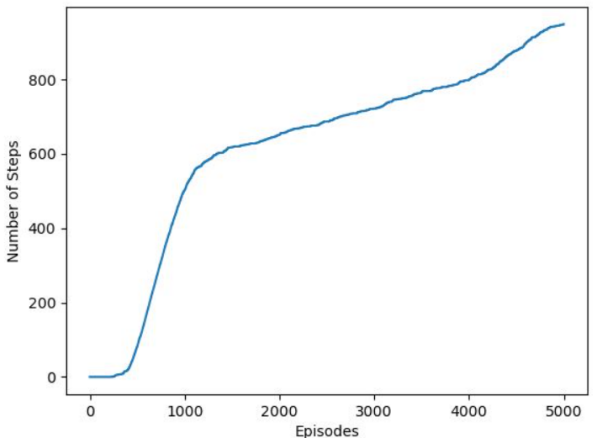

Figure 16: Prioritized Sweeping experimental results

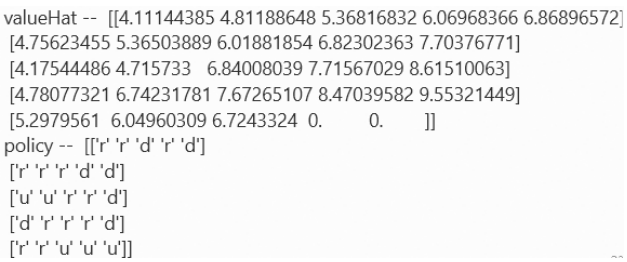

Figure 17: Value function