Semi gradient N Step SARSA and REINFORCE with Baseline Project

Aparajith Raghuvir and Kartikay Gupta

December 2023

1 Introduction

Kartikay Gupta worked on REINFORCE Baseline Algorithm and Aparajith worked on Semigradient N Step SARSA. Development of code, hyperparameter tuning and report writing were done independently and in parallel.

1.1 Methods

1.1.1 Episodic Semi-Gradient N-Step SARSA Algorithm

The episodic semi-gradient N-step SARSA algorithm is a two-step extension of SARSA. The first component involves N-step bootstrapping. N-step SARSA extends SARSA by considering a sequence of N steps into the future. It computes the target Q-value in the temporal difference update by bootstrapping from the N-step rewards and the Q-value.

The second component involves function approximation techniques. When using grids and tables of Q-values becomes impractical, function approximation techniques such as linear approximation and neural networks may be employed. Alternatively, constants and grids could be used, rendering the semi-gradient unnecessary in the update equation for the N-step semi-gradient SARSA.

The advantages over SARSA are manifold, ranging from more stable learning to increased efficiency due to reduced variance and a more effective utilization of collected data samples.

Algorithmic Pseudocode The pseudo-code that follows is from Sutton and Barto's Reinforcement Learning

Algorithm 1 Episodic semi-gradient n-step SARSA

Require: A differentiable action-value function parameterization $\hat{q} : S \times A \times R^d \to R$

Require: A policy π (if estimating q^π)

Algorithm parameters: step size $\alpha > 0$, small $\epsilon > 0$, a positive integer n
Initialize value-function weights $w \in R^d$ arbitrarily (e.g., $w = 0$)
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 $A_0 \leftarrow \pi(\cdot|S_0)$ or ϵ-greedy w.r.t. $\hat{q}(S_0, \cdot, w)$
    $T \leftarrow \infty$
    while $t < T$ **do**
        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 $A_{t+1} \leftarrow \pi(\cdot|S_{t+1})$ or ϵ-greedy w.r.t. $\hat{q}(S_{t+1}, \cdot, w)$
        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**
                $G \leftarrow G + (\gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, w))$
                $w \leftarrow w + \alpha[G - \hat{q}(S_\tau, A_\tau, w)]\nabla_w \hat{q}(S_\tau, A_\tau, w)$
            end if
        end if
        $t \leftarrow t + 1$
    end while
end for

2 Semi Gradient N Step SARSA

2.1 MDP 1 - 687 Gridworld

The first experiment is with 687-Gridworld, the same domain as was described in HW3. A simple linear function was used to approximate the value of each state and the hyperparameters to be optimized were N which is how many steps ahead we bootstrap and update based of, the learning rate α, and the exploration parameter ϵ.

This was a simplistic demonstration of n step semi gradient sarsa and was more to test and figure out whether the algorithmic implementation was correct, and it was. It is not expected to require any significant hyperparameter tuning, and indeed did not require the same. N value of 4 seemed to work well, and it was indeed possible to try out every feasible N from 1 to 25 given that the 687 gridworld is a 5x5 grid. For the learning rate, a value of 0.1 was used and this was derived empirically by experimenting with different rates such as 0.1, 0.01 and so on. It was also thought about theoretically which values would work well given that we know what the true value function of gridworld is, and an learning rate of 0.1 seemed appropriate for this experiment.

Learning curves can be seen below for the hyper-parameters α being 0.1, ϵ being 0.8, N set to 4 over 200 to 1000 episodes, and it is clearly visible that the agent is indeed learning over time, but not quite converging to the true value function as evidenced by the mean squared error curve, but this is expected behaviour.

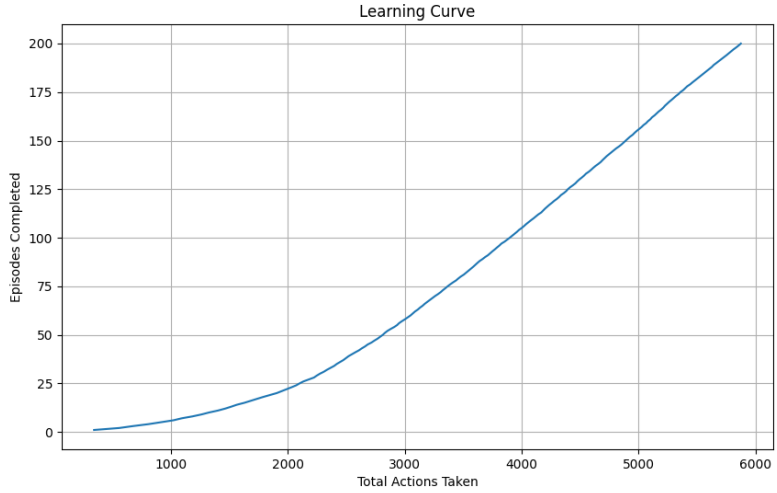

Figure 1: Learning curve after 200 episodes

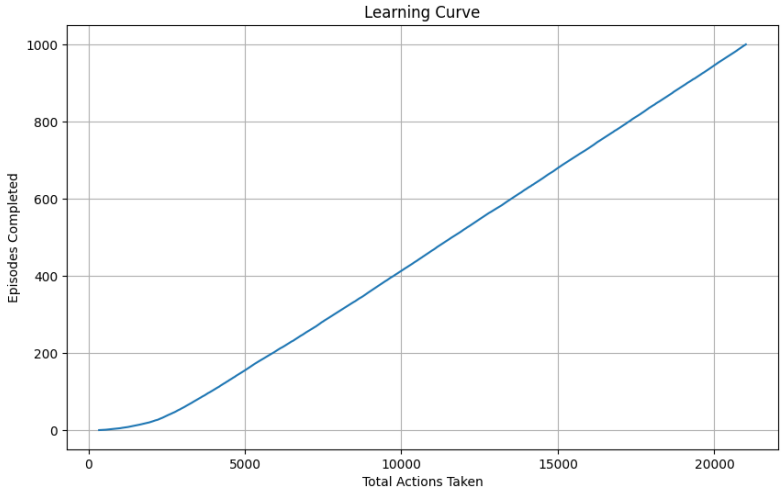

Figure 2: Learning curve after 1000 episodes

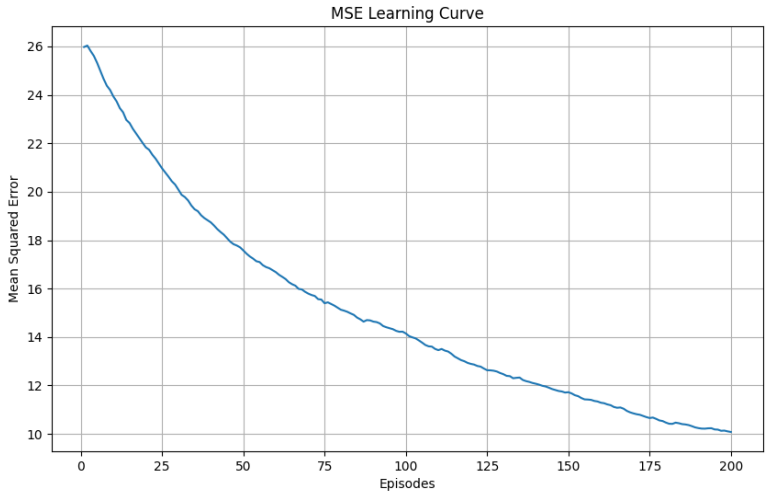

Figure 3: MSE curve after 200 episodes

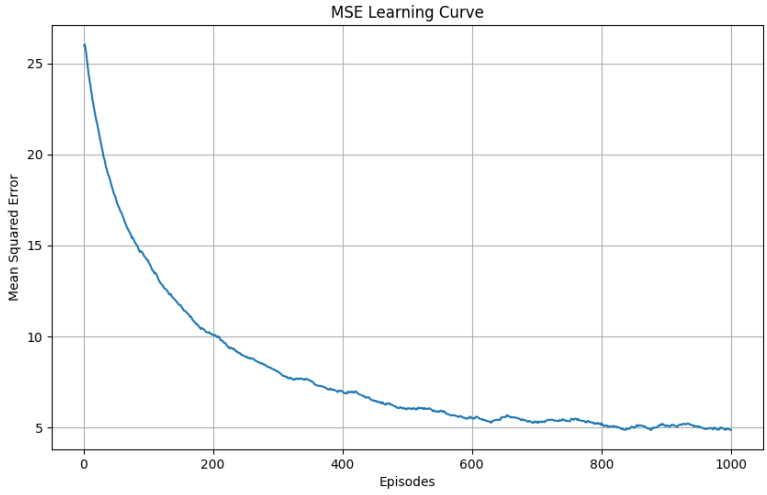

Figure 4: MSE curve after 1000 episodes

2.2 Mountain Car

The second MDP experimented with was the mountain car. The environment and MDP is exactly as described in the RL book. The task here is to drive

an underpowered car up a mountain road. Gravity here is much more powerful than the car's engine so it cannot just accelerate upwards. Intuitively, we know that the best way to go up the steep mountain slope is to accelerate backwards up a hill, and then quickly come downwards and use our forward momentum to push ourselves up the hill. This is what the agent needs to learn.

The reward here is -1 until the car climbs and reaches its destination. The agent is allowed to choose between three possible actions - full throttle forward, full throttle backwards and neutral. The car moves based on the following simplistic equations.

Its position, x_t, and velocity, $\dot{x}_t$, are updated by

$$\begin{aligned} x_{t+1} &= \text{bound}\left(x_t + \dot{x}_{t+1}\right) \\ \dot{x}_{t+1} &= \text{bound}\left(\dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)\right), \end{aligned}$$

and the bound is the following: $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$.

Further, as is intuitively obvious, when the left sided extreme position is reached, the velocity is set to zero and when the right sided extreme position is reached, the agent has reached the goal state and the episode is terminated.

The state values here are clearly continuous, so grid tilings were used and 9 tiles were used with 1/8th of the bound being covered by each and these were used linearly to compute the state-value function. The tile coding software developed by Sutton and Barto and made available open source was used as part of this project.

The hyperparameters that needed to be tuned here were N, ϵ for how greedy we want the action selection to be, α, the learning rate and the number of tilings. The tilings were experimented in powers of 2 and it seemed to work fine with 8, and this made sense intuitively as the state space was not that large and with just three possible actions, this was sufficient granularity for the agent to learn.

Extensive exploration was forced by setting a high exploration parameter value of 0.5 and after some experimentation by just searching through different ϵ values like through a grid, based on HW2 where I tuned alpha the learning rate, used some carry over intuition and also over a grid, and with an alpha of 0.06, learning seemed to work best.

N values of both 4 and 8 seemed to work well, as indeed did one, but it was faster learning with a higher N value and this is expected behavior. Learning curves can be seen below.

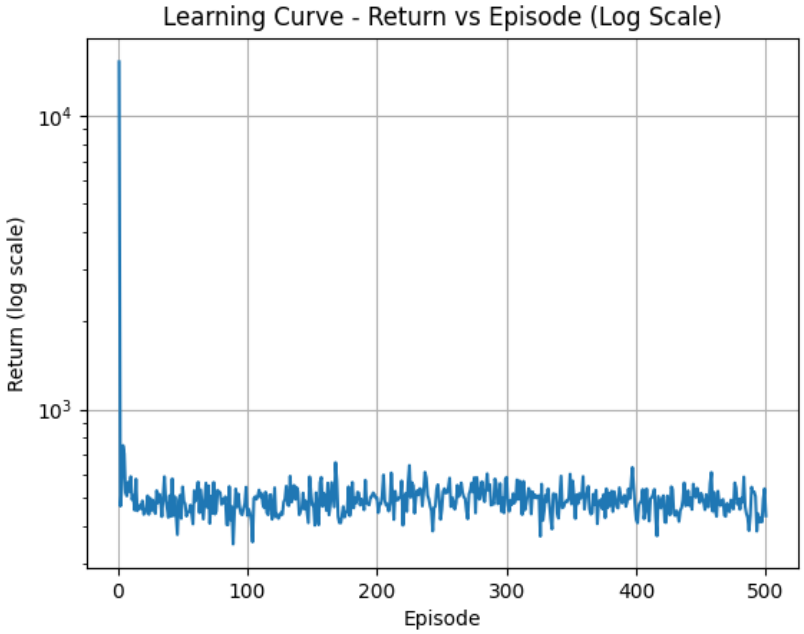

Figure 5: Learning curve for N=8, alpha=0.06

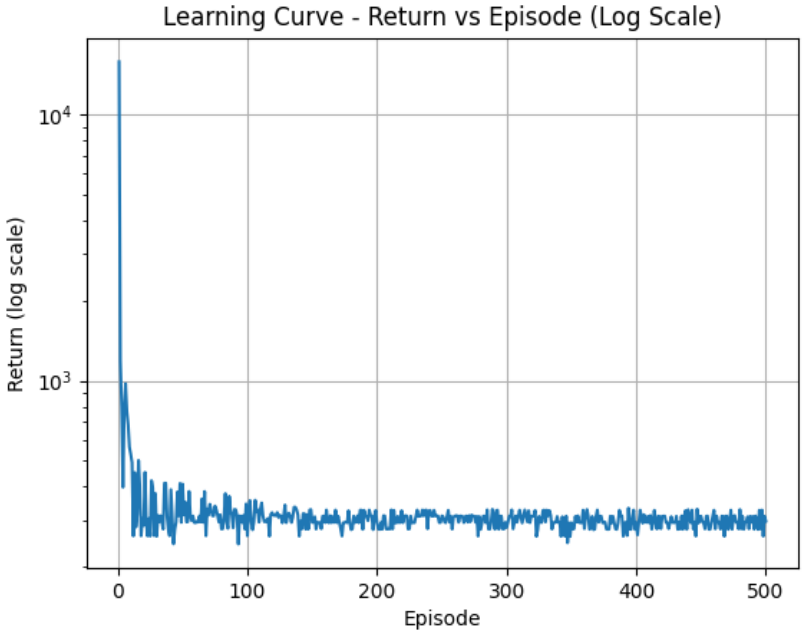

Figure 6: Learning curve for N=1, alpha=0.04

3 REINFORCE Algorithm with a Baseline:

- REINFORCE is a policy gradient method used for updating the policy in RL based on the gradient of expected rewards.
- The inclusion of a baseline in REINFORCE helps in reducing the variance of gradient estimates by subtracting a baseline value from the returns.

Hyperparameters for REINFORCE Algorithm:

Algorithm 2 REINFORCE with Baseline (episodic) for estimating $\pi^*, \pi, \hat{v} \approx v^*$

Require: A differentiable policy parameterization $\pi(a|s, \theta)$
Require: A differentiable state-value function parameterization $\hat{v}(s, w)$
Algorithm parameters: step sizes $\alpha_\theta > 0$, $\alpha_w > 0$
Initialize policy parameter $\theta \in R^{d_\theta}$ and state-value weights $w \in R^d$ (e.g., to 0)
while True (for each episode) **do**
    Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$
    for $t = 0, 1, ..., T-1$ (for each step of the episode) **do**
        $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (where G_t is the return)
        $\delta = G_t - \hat{v}(S_t, w)$
        $w \leftarrow w + \alpha_w \delta \nabla_w \hat{v}(S_t, w)$
        $\theta \leftarrow \theta + \alpha_\theta \cdot G_t \cdot \nabla_\theta \ln \pi(A_t|S_t, \theta)$
    end for
end while

- **Discount Factors (*discount_factors*):**
  - A lower discount factor gives more weight to immediate rewards, while a higher one considers future rewards more significantly.
  - For experimentation: Trying different values like 0.95, 0.99, and 0.999 can help understand their impact on learning.
- **Number of Episodes (*num_episodes_list*):**
  - More episodes allow the agent to explore the environment extensively, potentially improving learning.
  - For experimentation: Trying different episode counts like 500, 1000, and 1500 evaluates how learning varies with these durations.
- **Maximum Steps per Episode (*max_steps_list*):**

– Affects how much exploration or exploitation an agent can perform within an episode.
– For experimentation: Trying various step limits such as 500, 1000, and 1500 examines their influence on learning performance.

Result: We conducted experiments to optimize hyperparameters, achieving a significant milestone where the agent attained a reward of 500 in cartpole.

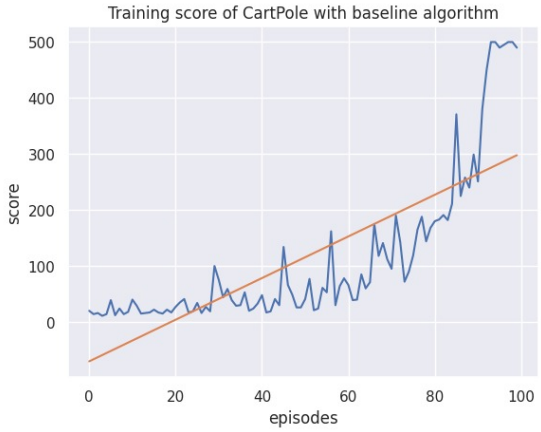

Figure 7: Cartpole in reinforce with baseline for single values of hyperparameter

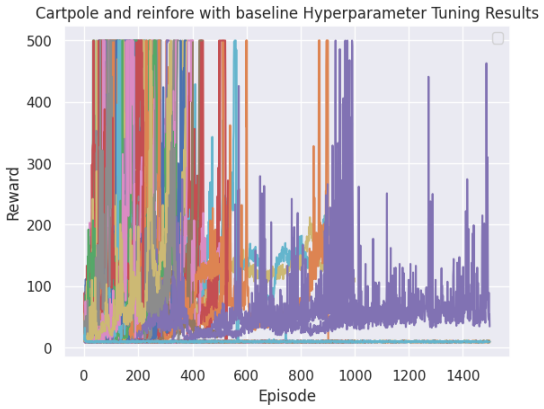

Figure 8: Cartpole in reinforce with baseline with hyper-parameter tuning

In the learning curve above this is the values used to get to the reward of 500 in cartpole at the earliest number of episodes discount factor: 0.95, num episodes: 100, max steps: 500, solved score: 210

Hyperparameters for 687-Grid World RL:

- **Learning Rates (*learning_rates*):**
  - Values like 0.01, 0.05, and 0.1 represent different magnitudes of adjustments made by the agent based on observed rewards.
- **Gamma Values (*gamma_values*):**
  - Values such as 0.9, 0.95, and 0.99 indicate the extent to which future rewards are discounted in the agent's decision-making process.
- **Sigma Values (*sigma_values*):**
  - Values like 0.1, 0.5, and 1.0 represent the degree of randomness or exploration the agent employs during learning.
- **Alpha Values (*alpha_values*):**
  - Values such as 0.01, 0.05, and 0.1 represent different scales of adjustment made to the agent's policy parameters or value estimates.

Result: We conducted experiments to optimize hyper-parameters, achieving a significant milestone where the agent attained a reward of 10 in 687 gridworld.

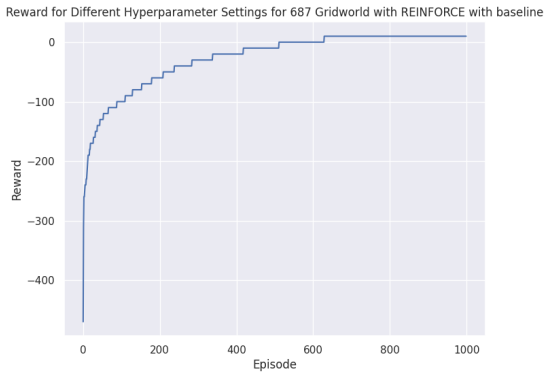

Figure 9: Reinforce Baseline 687 with hyper-parameter

Hyperparameters for Mountain Car RL:

- **Learning Rates (*step_sizes_thetas*):**
  - Values like 0.001, 0.01, and 0.1 represent different magnitudes of adjustments made by the agent based on observed rewards.
- **Gamma Values (*gamma*):**
  - Values such as 0.9, 0.95, and 0.99 indicate the extent to which future rewards are discounted in the agent's decision-making process.

- **Sigma Values (*step_size_omegas*):**

  .Values like 0.05, 0.1, and 0.2 represent the degree of randomness or exploration the agent employs during learning.

- **Alpha Values (*num_of_tilings_values*):**

  .Values such as 4, 8, and 16 represent different scales of the number of tilings used by the agent in its learning process.
  Result: Agent is not getting the positive reward so thts why it is truncating at 200.

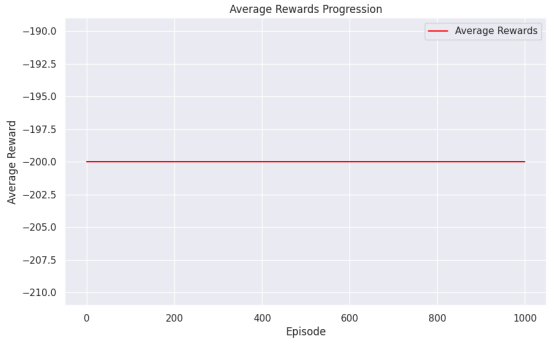

Figure 10: Reinforce Baseline mountain$_car without hyperparameter$

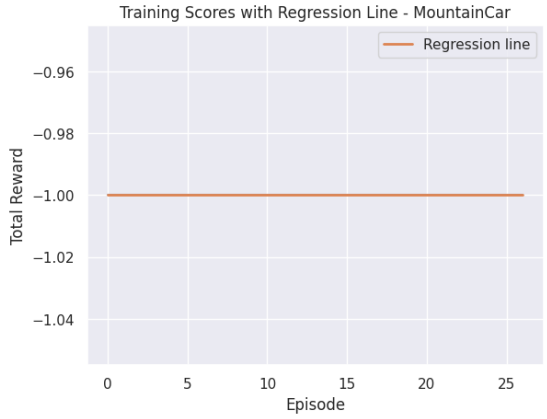

Figure 11: Reinforce Baseline mountain car with hyperparameter

References

– Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, 2nd edition, The MIT Press, 2018.