IMPLEMENTING TRUE ONLINE SARSA LAMBDA AND ONE STEP ACTOR CRITIC ON VARIOUS ENVIRONMENTS

Aishwarya Nair

1 TRUE ONLINE SARSA (λ)

This section was implemented and compiled by Aishwarya Nair

1.1 ENVIRONMENTS CHOSEN:

The algorithm is evaluated on the following domains:

- 687-Gridworld
- Cartpole
- Mountain car

The definitions for the Cartpole and Mountain Car environments are taken from Open AI gymnasium **?**. The implementation however is done from scratch.

The 687-Gridworld is implemented according to the definition as mentioned in Reinforcement Learning 2nd edition da Silva (2023).

1.2 687 GRIDWORLD

Implementing the True Online Sarsa (λ) as is on the 687-Gridworld was quite a bit of challenge due to the fact that the algorithm as mentioned in every source is for a continuous state space. After experimenting with multiple approaches including One Hot Encoded Vector State Space representation, the idea that I stumbled upon was to derive a tabular version of the True Online Sarsa (λ) algorithm using the tabular version of the True Online TD (λ) algorithm from van Seijen et al. (2015). The psueudocode Algorithm 1 for the same is derived from the paper mentioned previously.

1.2.1 METHODS USED:

- $\_\_init\_\_$: Initializes the parameters including α, γ, λ and the Q table and the eligibility traces e.
- get_q_values: Takes input of state and action and returns the q value
- reset: Resets the values of $\alpha, \gamma, q\ or\ w, e$
- $update\_q$ updates the q table and returns the value of δ
- $calculate\_values$ calculates the value function of the tabular policy
- $calculate\_variance$ calculates the variance from the optimal function the tabular policy
- $epsilon\_greedy$ returns the next action to be taken based on the value of epsilon
- $optimal\_policy$ obtains the optimal policy for a tabular function
- $print\_matrix$ prints the optimal policy
- run: implements an episode of the algorithm
- run episodes: takes an input integer x and runs x episodes
- d_0: returns the starting state for the MDP
- transition: located in the env files, returns the next state

- reward: located in the env files, returns the reward
- terminal: located in the env files, checks if state is terminal or not

1.2.2 Pseudocode:

Algorithm 1 Tabular True Online Sarsa (λ)

```
Initialize:
    $q(s, a) \leftarrow 0$ for all $s, a$
loop over episodes:
    Initialize:
    s $\sim d_0$
    $e(s) \leftarrow 0$ for all S
    $q_{old} \leftarrow 0$
    while $s$ is not terminal do
        $s' \leftarrow env.transition(s, a)$
        $a' \leftarrow epsilon\_greedy(s)$
        $\Delta Q \leftarrow q(s, a) - q_{old}$
        $q_{prime} \leftarrow q(s', a')$ with $get\_q$
        $q_{old} \leftarrow q_{prime}$
        $\delta \leftarrow update\_q(s, a)$
        $e(s) \leftarrow (1 - \alpha)e(s) + 1$
        loopover all s, a:
            $q \leftarrow q + \alpha(\delta + \Delta Q) * e(s)$
            $e(s) \leftarrow \gamma\lambda e(s)$
        end loop
        $q(s, a) \leftarrow q(s, a) - \alpha\Delta Q$
        $s \leftarrow s'$, $a \leftarrow a'$
    end while
end loop
```

Now, this algorithm may not be completely correct theoretically, but by intuition and from a practical viewpoint it works. It provides a decent estimate of the value function and the optimal policy. The hyperparameters were tuned using grid search, however it was fairly simple to find a set of optimal hyperparameters, owing to the simplicity of the domain.

The optimal hyperparameters as obtained by random search is:

$$\begin{aligned} \alpha &= 0.3 \\ \gamma &= 0.8 \\ \lambda &= 0.4 \\ iters &= 1000 \\ \epsilon &= 0.6[decaying] \end{aligned}$$

The variance per episode and the number of steps taken per episode are shown in figures 1 2 respectively.

This implememtation is comparable to the offline version of Sarsa (λ). The graphs are fairly comparable. On comparison of figure 2 which is the online version of sarsa lambda with the offline version of sarsa [figure 3], we can see that the minima of the online version is achieved faster as compared to the offline version of the algorithm.

1.3 Cartpole

Implementing Cartpole was slightly more challenging than a tabular state space representation, mainly due to hyperparameter tuning and weight overflows. Hyperparameter tuning in this case

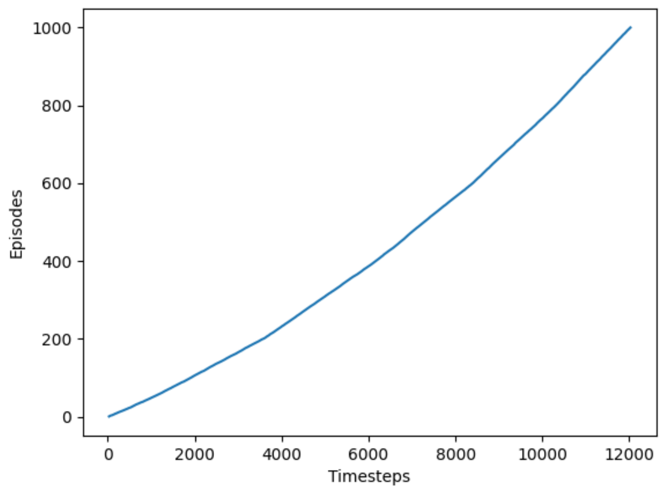

Figure 1: number of episodes versus steps taken for 687-Gridworld with true online sarsa lambda

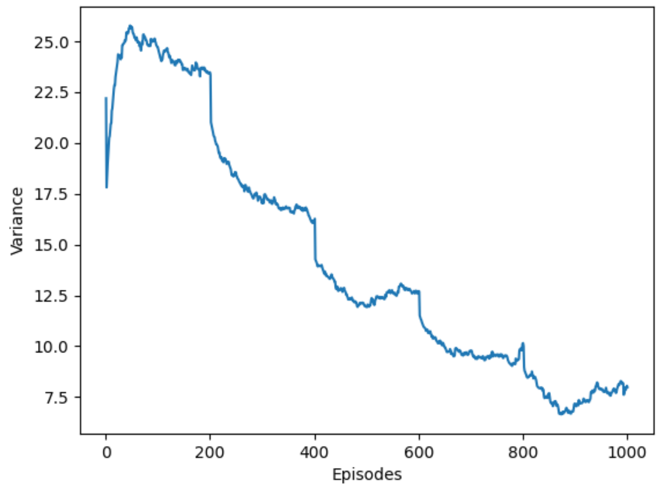

Figure 2: Variance versus number of episodes for 687-Gridworld with true online sarsa lambda

was carried out by random search, evaluating over a range of parameters, other alternatives like hyperopt library was also tried, but due to time constraints, using a range and evaluating over the range of parameters proved to be the most efficient. Most of the functions are similar to the ones used in 687-Gridworld. The new functions added include

- $fourier\_series$: takes input as state and returns the fourier basis for that state
- normalize: normalizes the state values

The pseudocode [Algorithm 2] for the algorithm is derived from van Seijen et al. (2015):

The weight overflow problem occurs when the learning rate is too high, causing the weights to increase exponentially. It is similar to the problem occurring in classical ML problems where if the initial weight and the learning rate are too high, the weight rises up to infinity.

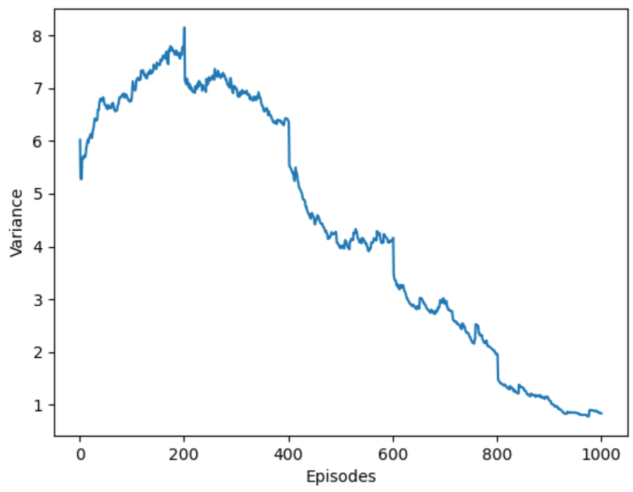

Figure 3: Variance versus number of episodes for 687-Gridworld with offline sarsa lambda

Algorithm 2 True Online Sarsa (λ)

```
Initialize:
    $\theta \leftarrow 0$
loop over episodes:
    Initialize:
    S $\sim d_0$
    $e(s) \leftarrow 0$ for all S
    $A \leftarrow epsilon\_greedy(s)$
    $q_{old} \leftarrow 0$
    $\Psi \leftarrow FourierBasis(s)$
    while $s$ is not terminal do
        obtain next state $S'$ and reward $r$ and next action $A'$ using $\epsilon$-greedy policy
        $\Psi' \leftarrow FourierBasis(S')$
        $q \leftarrow \theta[A]^T\psi$
        $\Delta Q \leftarrow q - q_old$
        $q_{old} \leftarrow q_{prime}$
        $\delta \leftarrow update\_q(s, a)$
        $e \leftarrow \gamma\lambda e + \psi - \alpha\gamma\lambda(e^T\psi)\psi$
        $\theta \leftarrow \theta + \alpha(\delta + \Delta Q)e - \alpha\Delta Q\psi$
        $q_{old} \leftarrow q_{prime}$
        $\psi \leftarrow \psi'$
        $A \leftarrow A'$
    end while
end loop
```

The learning curves for the hyperparams are shown in figure 4 and their learning curves with the standard deviation are shown in figure 6

After trying out a bunch of learning rates and weight initializations, we get the optimal hyperparameters as hyperparameter number 5 [purple line in both graph]. The legends have been mixed up due

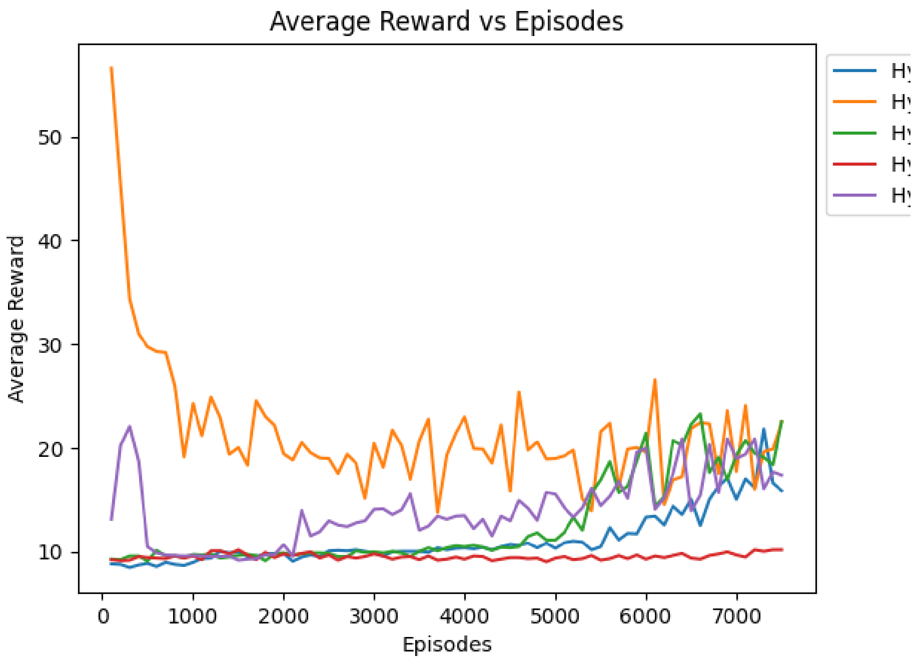

Figure 4: Cartpole average reward

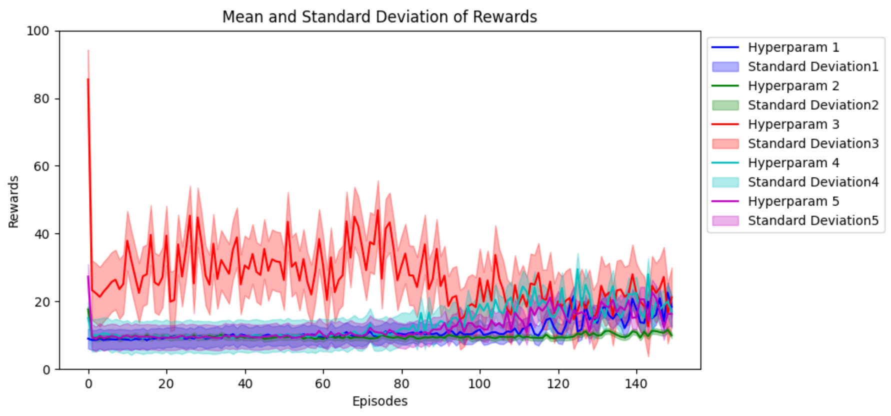

Figure 5: Cartpole Learning curve with standard deviation

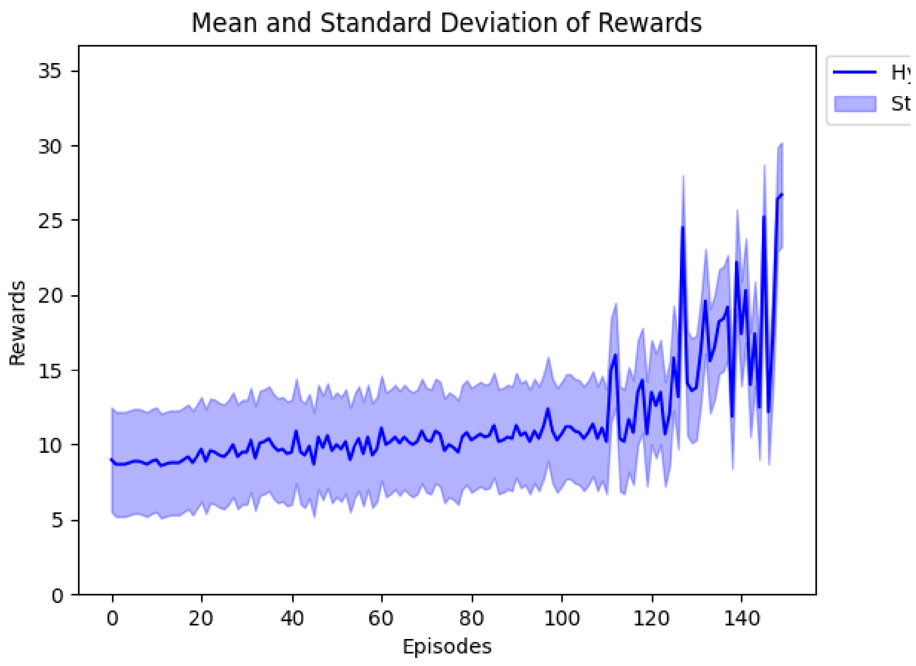

Figure 6: Cartpole Learning curve with standard deviation for best hyperparam

to technical glitches, but the shape of the curve tells us most of the information required:

$$\alpha = 0.01$$
$$\gamma = 0.9$$
$$\lambda = 0.4$$
$$iters = 1500$$
$$\epsilon = 0$$
$$degree\ of\ fourier\ basis = 2$$

Other hyperparameters tried include:

Hyperparameter 1 [blue line in Fig 4, blue line in fig 6]::

$$\alpha = 0.025$$
$$\gamma = 0.4$$
$$\lambda = 0.4$$
$$iters = 1500$$
$$\epsilon = 0$$
$$degree\ of\ fourier\ basis = 2$$

Hyperparameter 2 [yellow line in Fig 4, red line in fig 6]:

$$\begin{aligned} \alpha &= 0.01 \\ \gamma &= 0.2 \\ \lambda &= 0.2 \\ iters &= 1500 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 2 \end{aligned}$$

The reason why this hyperparameter did not work is due to a low gamma, prioritizing short term goals for long term rewards leading to fluctuating returns.

There are signs of learning, however the learning is very slow.

Hyperparameter 3 [green line in Fig 4, green line in fig 6]:

$$\begin{aligned} \alpha &= 0.05 \\ \gamma &= 0.6 \\ \lambda &= 0.6 \\ iters &= 1500 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 2 \end{aligned}$$

This hyperparam seems to work the best but it still prioritizes short term rewards over long term goals.

Hyperparameter 4 [dark blue lines in both the graphs]:

$$\begin{aligned} \alpha &= 0.1 \\ \gamma &= 0.8 \\ \lambda &= 0.8 \\ iters &= 1500 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 2 \end{aligned}$$

This hyperparam was very close to the learning curve needed. Hence the final parameters were chosen to ensure that the MDP is not myopic. The learning curve is as shown in figure 5

The exploration parameters were chosen as zero. For domains with continuous state space representation expressed in terms of Fourier basis, the exploration parameter causes zero or very minor fluctuations in the optimal policy in my experience.

1.4 Mountain Car

Mountain car is a deceptively tough problem, using the definiton of OpenAI (2023), we know that the environment awards a -1 for every step taken. The environment considers every action as bad until it actually reaches the goal state. Intuitively, it is as if we consider every action taken in our life as bad. Hence even some DNN's fail to optimize this problem. Reward shaping can fix this problem, however for now, it was not required.

Secondly, like cartpole it also has the problem of exploration. There are many strategies to fix this problem, something like novelty search. However, even though ineffective, the true online sarsa (λ) does work.

The pseudocode used was the same as the cartpole algorithm. The only minor change in the environment compared to the OpenAI (2023) algorithm is that the algorithm terminates after 1000-5000

steps. The optimal hyperparameters found were:
Hyperparameter 1:

$$\begin{aligned} \alpha &= 0.05 \\ \gamma &= 0.9 \\ \lambda &= 0.4 \\ iters &= 1000 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 3 \end{aligned}$$

Some other hyperparameters very close to optimality found during grid search are: Hyperparameter 2:

$$\begin{aligned} \alpha &= 0.01 \\ \gamma &= 0.9 \\ \lambda &= 0.4 \\ iters &= 1500 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 3 \end{aligned}$$

The above hyperparameter is very close to optimality, however with lambda being 0.8 it does not prioritize long term rewards as much, however during the rendering of the mountain car using the gym library, one could see that it was trying to reach the goal state by generating the kinetic energy by going back up the slope and accelerating down, but not enough. Hence a higher lambda was required. Hyperparameter 3:

$$\begin{aligned} \alpha &= 0.025 \\ \gamma &= 0.85 \\ \lambda &= 0.4 \\ iters &= 1000 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 3 \end{aligned}$$

This parameter being so close to the previous one shows a very similar curve to the previous one. Hyperparameter 4:

$$\begin{aligned} \alpha &= 0.03 \\ \gamma &= 0.9 \\ \lambda &= 0.4 \\ iters &= 1000 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 3 \end{aligned}$$

The learning rate for the above hyperparameter was too high, resulting into the model behaving erratically with a very high standard deviation.

Hyperparameter 5:

$$\begin{aligned} \alpha &= 0.001 \\ \gamma &= 0.99 \\ \lambda &= 0.4 \\ iters &= 1000 \\ \epsilon &= 0 \\ degree\ of\ fourier\ basis &= 2 \end{aligned}$$

Due to the learning rate being too low, the agent did not learn anything except for the times it got lucky with the starting states.

The learning curves for the hyperparams are shown in figure 9 with the averages and standard deviation of the learning curve, in figure 7 with the average of the learning curve and in figure 8 for the best hyperparameter.

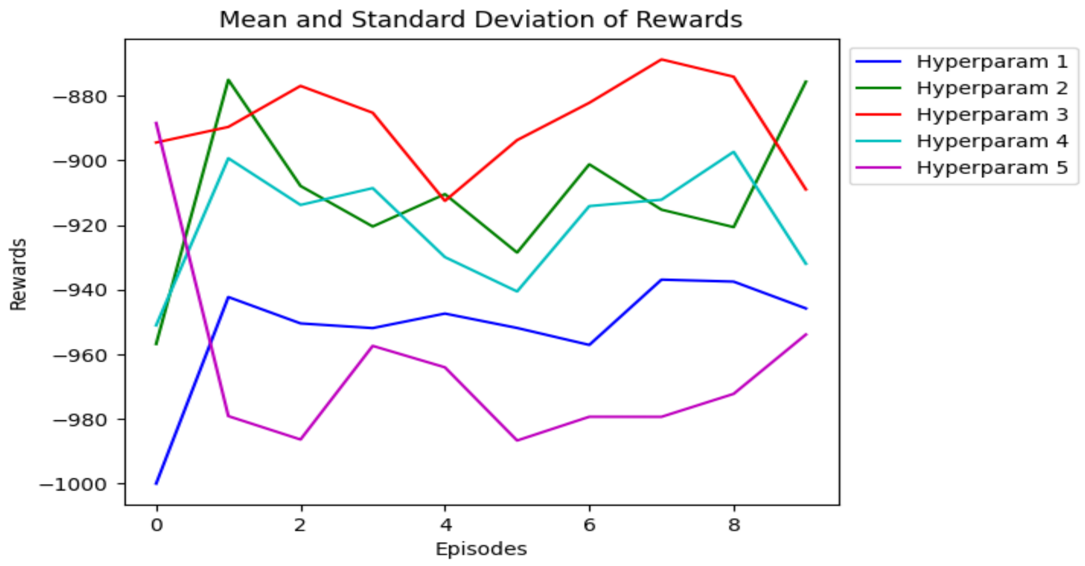

Figure 7:

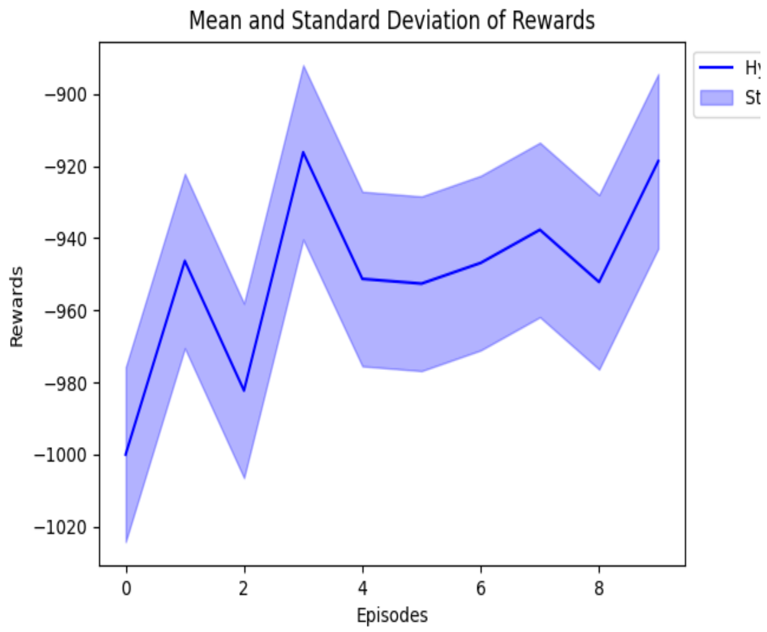

Figure 8: Best hyperparameter found for mountain car

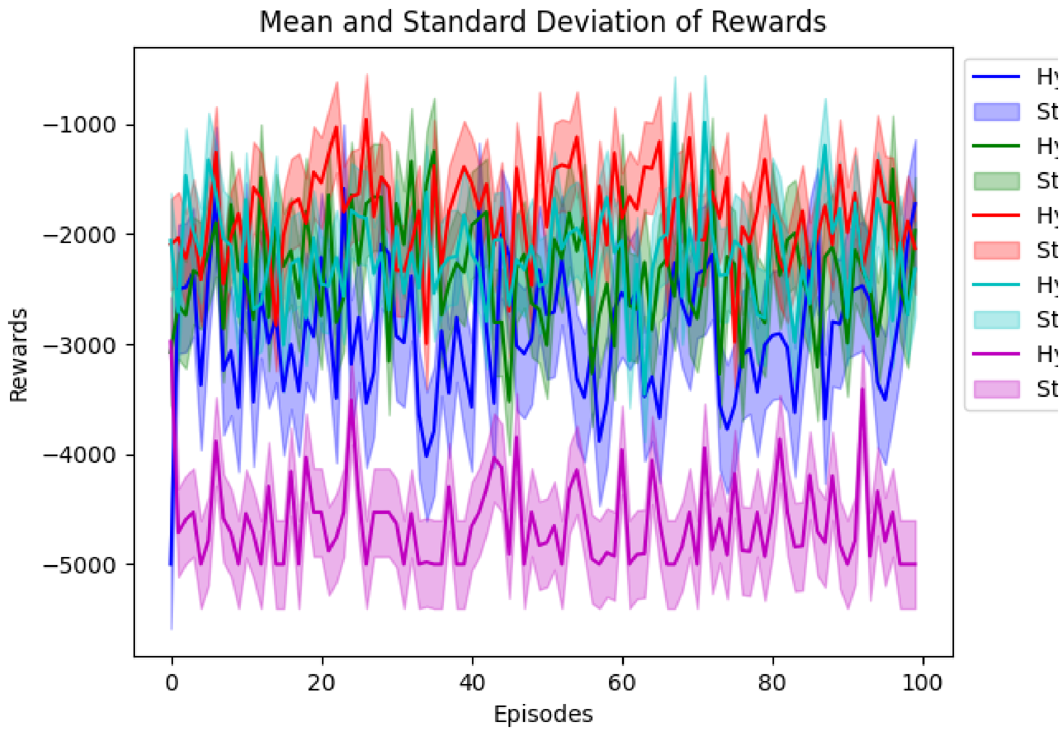

Figure 9: Mountain Car rewards with mean and standard deviation

REFERENCES

Prof. Bruno C. da Silva. Lecture notes - fall 2023, 2023. URL `https://people.cs.umass.edu/~bsilva/courses/CMPSCI_687/Fall2023/Lecture_Notes_v1.0_687_F23.pdf`.

OpenAI. Openai, 2023. URL `https://gymnasium.farama.org/`.

Harm van Seijen, Ashique Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, and Richard S. Sutton. True online temporal-difference learning. CoRR, abs/1512.04087, 2015. URL `http://arxiv.org/abs/1512.04087`.