DA6400 - Reinforcement Learning

Programming Assignment 1

Patel Shrey Rakeshkumar (ME21B138)
Param Jivrajani (PH21B006)

Table of Contents

1 Introduction

This report consists of using OpenAI's Gym library to simulate multiple environments using SARSA and Q-Learning algorithms with ϵ-greedy and Softmax policies, respectively.

The environments simulated in this assignment are :

- **CartPole-v1**
- **MountainCar-v0**

The environments were simulated over a range of various hyperparameter values which was done using *wandb*.

2 Environments

2.1 Environment 1 : CartPole-v1

CartPole-v1 is one of the most widely used environments in gymnasium. The environment consists of the pole attached to a cart moving along a frictionless track. The goal is to balance the pole on the cart for as long as possible, applying left and right forces.

2.1.1 Description of the Environment

- **Action Space** : Consists of an ndarray of shape (1,) which can take the values {0,1}. The actions are encoded as follows :
  - 0 : Push cart to the left
  - 1 : Push cart to the right
- **Observation Space** : Consists of an ndarray of shape (4,) with the following values :

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad ($-24°$)	~ 0.418 rad ($24°$)
3	Pole Angular Velocity	-Inf	Inf

Table 1: Observation Limits

The cart x-position (index 0) can be take values between (-4.8, 4.8), but the episode terminates if the cart leaves the (-2.4, 2.4) range.

The pole angle can be observed between (-0.418, 0.418) radians (or 24°), but the episode terminates if the pole angle is not in the range (-0.2095, 0.2095) (or $\pm 12°$)

- **Rewards** : A default reward of +1 is given for each step since the goal is to balance the pole for as long as possible.

- **Episode End** : The episode either by truncation after completion of the maximum steps, which is 500 for the CartPole-v1 environment, or by termination if any of the states crosses the threshold value given above.
- **Starting State** : All observations are assigned a uniformly random value in [-0.05, 0.05].

2.2 Environment 2 : MountainCar-v0

Mountain Car is a deterministic MDP with the version v0 consisting of a discrete action space. A car is placed stochastically at the bottom of a sinusoidal valley with the goal of reaching the goal state at the top on the right hill, by accelerating the car.

2.2.1 Description of the Environment

- **Action Space** : Consists of an ndarray of shape (1,) which can take the following values :
  - 0 : Accelerate to the left
  - 1 : Don't accelerate
  - 2 : Accelerate to the right
- **Observation Space** : Termination occurs if position of the car is greater than or equal to 0.5

Num	Observation	Min	Max	Unit
0	position of the car along the x-axis	-1.2	0.6	position (m)
1	velocity of the car	-0.07	0.07	velocity (v)

Table 2: Observations of the car's position and velocity

- **Reward** : The goal is to reach the flag placed on top of the right hill as quickly as possible, as such the agent is penalized with a reward of -1 for each timestep.
- **Episode End** : The episode was terminated after completion of the maximum steps, which is 200 for the MountainCar-v0 environment, or by truncation if any of the states crossed the threshold value given above.
- **Starting State** : The position of the car is assigned a uniform random value in [-0.6 ,-0.4]. The starting velocity of the car is always assigned to 0.

3 Implementation and Code Snippets

The implementation was done using the inbuilt ***gymnasium*** library in Python.

- The state space in the above environments are continuous with a Discrete Action space.
- The state space was discretized using the *np.discretize* function, passing the **number of bins** as a hyperparameter.

  The code implementation is given below for the CartPole-v1 case. The implementation for MountainCar-v0 is similar other than the limits :

```
class discretization:
    def __init__(self, env_name, bin_step_list):
        self.env_name = env_name
        if env_name == 'CartPole-v1':
            self.POS_BINS = np.linspace(-4.8, 4.8, num=bin_step_list[0],
                                                    endpoint=False)
            self.VEL_BINS = np.linspace(-4, 4, num=bin_step_list[1], endpoint=
                                                    False)
            self.ANG_BINS = np.linspace(-0.5, 0.5, num=bin_step_list[2],
                                                    endpoint=False)
            self.AV_BINS = np.linspace(-4, 4, num=bin_step_list[3], endpoint=
                                                    False)

    def discretize_state(self, state):

        if self.env_name == 'CartPole-v1':
            pos_idx = np.clip(np.digitize(state[0], self.POS_BINS) - 1, 0, len(
                                                    self.POS_BINS)-1)
            vel_idx = np.clip(np.digitize(state[1], self.VEL_BINS) - 1, 0, len(
                                                    self.VEL_BINS)-1)
            ang_idx = np.clip(np.digitize(state[2], self.ANG_BINS) - 1, 0, len(
                                                    self.ANG_BINS)-1)
            av_idx = np.clip(np.digitize(state[3], self.AV_BINS) - 1, 0, len(
                                                    self.AV_BINS)-1)
            return pos_idx, vel_idx, ang_idx, av_idx
```

- SARSA algorithm was implemented using the ϵ-greedy policy and Q-Learning was implemented using the softmax policy.

  The policies were implemented as follows :

```
def get_action(self, state_idx):
    if self.policy == 'epsilon_greedy':
        if np.random.random() < self.policy_param:
            return np.random.randint(0, self.num_actions)
        else:
            return np.argmax(self.Q[state_idx])

    elif self.policy == 'softmax':
        q_values = self.Q[state_idx] / self.policy_param
        q_values = q_values - np.max(q_values)
        exp_values = np.exp(q_values)
        probabilities = exp_values / np.sum(exp_values)
        return np.random.choice(self.num_actions, p=probabilities)
```

  The updates for each of them are shown in the code below:

```
def update(self, state_idx, action, reward, terminated, next_state_idx):
    if self.algo == 'Q-Learning':
        q_next = 0 if terminated else np.max(self.Q[next_state_idx])
        td_error = reward + self.discount_factor * q_next - self.Q[state_idx][
                                                action]
        self.Q[state_idx][action] += self.learning_rate * td_error

    elif self.algo == 'SARSA':
        next_action = self.get_action(next_state_idx)
        q_next = 0 if terminated else self.Q[next_state_idx][next_action]
```

```
        td_error = reward + self.discount_factor * q_next -
        self.Q[state_idx][action]
        self.Q[state_idx][action] += self.learning_rate * td_error
```

- The Q-table was maintained as a numpy array instead of a dictionary because of faster runtime.
- The Agent was run for 10000 episodes for CartPole-v1 and 5000 episodes for the MountainCar-v0 environment. The episodes were run till the condition for termination or truncation was reached.
- The total score was appended by adding the reward at each step of the episode. The score array was then averaged over 5 experiments with random seeds.
- The total steps were also maintained for each episode in a separate array. The total steps are equal to the total score for CartPole and negative of the score for MountainCar.
- The score and steps were averaged over 5 experiments with random seeds and the average value array over the 5 experiments was used for learning.

```
for i in range(num_expts = 5):
    agent = Agent(
        learning_rate=learning_rate,
        discount_factor=discount_factor,
        policy=policy,
        algo=algo,
        policy_param=param,
        param_decay=param_decay,
        final_param=final_param,
        num_actions=env.action_space.n,
        bin_step_list=bin_step_list
    )

    for episode in tqdm(range(n_episodes), desc=f"Experiment {i+1}/{num_expts}")
                                                    :
        obs, _ = env.reset()
        done = False
        score = 0
        steps = 0

        while not done:
            state_idx = dis.discretize_state(obs)
            action = agent.get_action(state_idx)
            next_obs, reward, terminated, truncated, _ = env.step(action)
            next_state_idx = dis.discretize_state(next_obs)

            agent.update(state_idx, action, reward, terminated, next_state_idx)

            score += reward
            steps += 1
            done = terminated or truncated
            obs = next_obs

        agent.decay_param()
        reward_avgs[i, episode] = score
        steps_avgs[i, episode] = steps
```

- The parameter value (ϵ for ϵ-greedy and temperature τ for softmax) was also passed as a hyperparameter.
- After each episode, the parameter value was decayed using a constant decay parameter which was also a hyperparameter.
- The plots shown in the report include the Total Reward averaged over the 5 experiments and the Rolling mean of the average reward.

  The rolling mean was calculated using the following code :

```
# Reward moving average
reward_moving_average = np.convolve(
        np.mean(reward_avgs, axis=0),
        np.ones(rolling_length) / rolling_length,
        mode="valid")

# Steps moving average
length_moving_average = np.convolve(
    np.mean(steps_avgs, axis=0),
    np.ones(rolling_length) / rolling_length,
    mode="valid")
```

4 Values used

The parameters used to obtain the results were :

Parameter	**Values**
Discount Factor	0.99 (fixed)
Algorithm	{SARSA, Q-Learning}
Policy	{ϵ-greedy, softmax}
Environment	{CartPole-v1, MountainCar-v0}

Table 3: Parameters used

The hyperparameters used in the code were:

Variables
Learning rate
Parameter value (ϵ / τ)
Decay rate value
Final parameter value (after decay)
Bin size for each state

Table 4: Hyperparameter Values

5 Results and Plots

The plots for each environment include:

- Comparative plot for the average reward and the rolling mean of the reward for the top 3 best hyperparameter values for each algorithm
- Comparative plot between the best hyperparameter value from SARSA and Q-Learning.
- The hyperparameters were optimized to obtain the minimum regret.
- The hyperparameters were obtained by performing a sweep on wandb within a range of values for each hyperparameter and using the **Bayes** method. The example code for the sweep is given below :

```
# Initialize sweep with data from .yaml file
with open("sweep.yaml", "r") as file:
sweep_configuration = yaml.safe_load(file)
wandb.require("core")
# Getting arguments for entity and project name
parser = argparse.ArgumentParser(description="Training script that
                                        return model weights")

parser.add_argument('-we','--wandb_entity',type=str,
            default='DA6400_PA1_param_shrey', help='Wandb Entity
            used to track experiments in the Weights & Biases dashboard')
parser.add_argument('-wp','--wandb_project',type=str,default='PA1',
            help='Project name used to track experiments in Weights & Biases
                                                    dashboard')
parser.add_argument('-c','--count',type = int,default=100,
help = 'Maximum number of seep per agent')
args = parser.parse_args()
sweep_id = wandb.sweep(sweep_configuration,entity=args.wandb_entity,
            project=args.wandb_project)
# Initilizing agent
wandb.agent(sweep_id, count=args.count)
```

- The *yaml* file defines the parameter range for each hyperparameter values
- The 3 hyperparameters with the least regret were chosen for comparison.

5.1 CartPole-v1

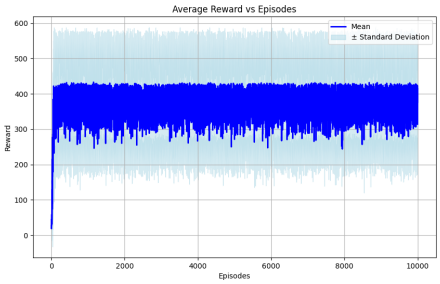
The CartPole-v1 was run for 500 steps per episode. The hyperparameters were tuned and the 3 best hyperparameter values with minimum regret are shown here.

5.1.1 Q-Learning with Softmax Policy

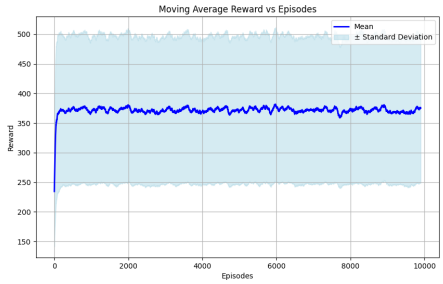
The 3 best parameter values are tabulated below :

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Bin size list	[9, 7, 10, 21]	Bin size list	[14, 10, 20, 13]	Bin size list	[12, 5, 22, 21]
No. of episodes	10000	No. of episodes	10000	No. of episodes	10000
No. of experiments	5	No. of experiments	5	No. of experiments	5
Learning rate	0.463	Learning rate	0.379	Learning rate	0.359
Discount factor	0.99	Discount factor	0.99	Discount factor	0.99
Policy	'softmax'	Policy	'softmax'	Policy	'softmax'
τ	4.296	τ	4.769	τ	3.132
Decay rate	0.989	Decay rate	0.998	Decay rate	0.998
τ_final	0.062	τ_final	0.064	τ_final	0.056

Table 5: Hyperparameter Values

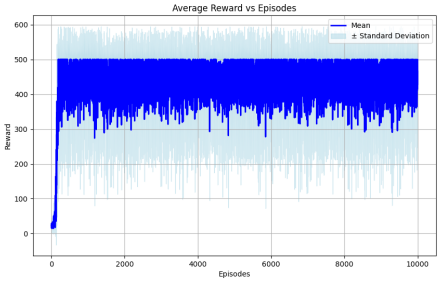


(a) Total Average Reward

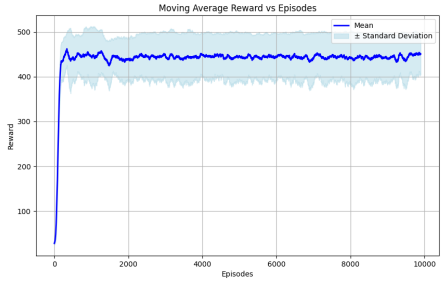


(b) Rolling mean reward

Figure 1: Learning Rate = 0.463 , Mean Regret = 129.49

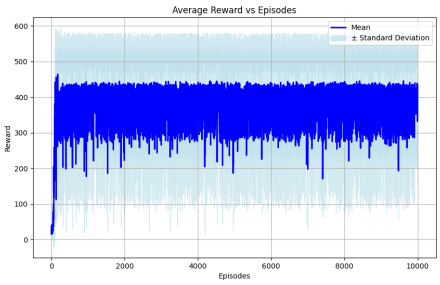


(a) Total Average Reward

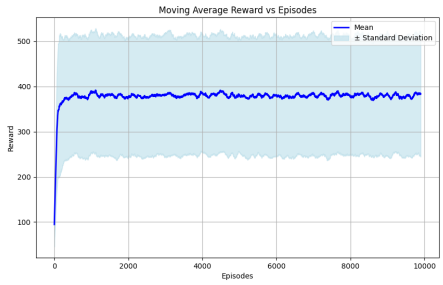


(b) Rolling mean reward

Figure 2: Learning Rate = 0.379 , Mean Regret = 62.37

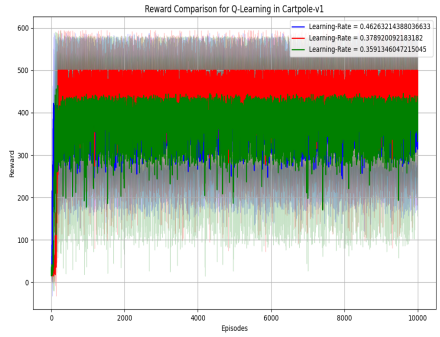


(a) Total Average Reward

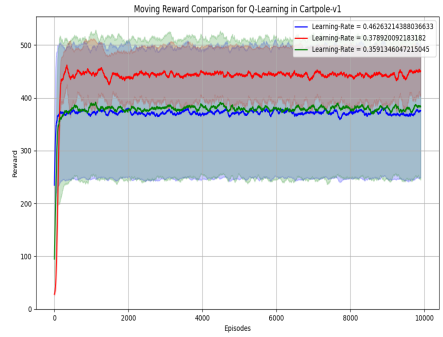


(b) Rolling mean reward

Figure 3: Learning Rate = 0.359 , Mean Regret = 123.62

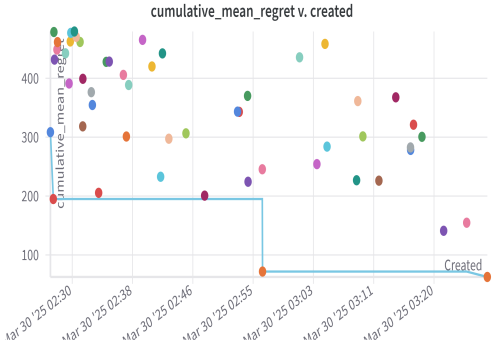


(a) Total Average Reward

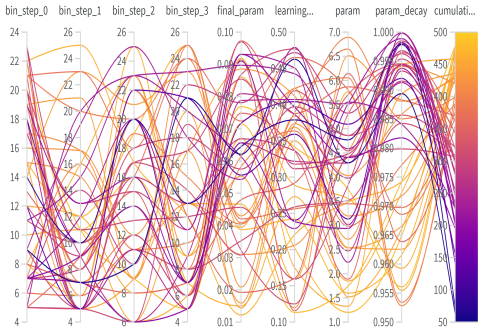


(b) Rolling mean reward

Figure 4: Comparison Plot



(a) Total Average Reward



(b) Rolling mean reward

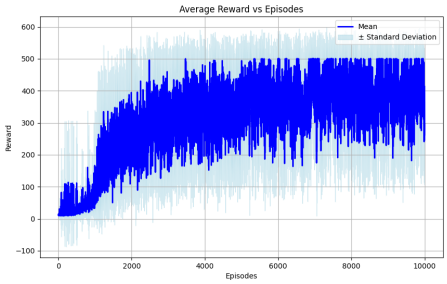
Figure 5: Wandb Plots - CartPole Q-Learning

5.1.2 SARSA with ϵ-greedy Policy

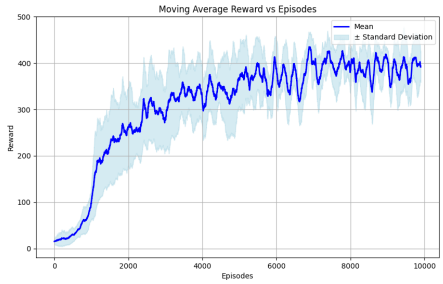
The 3 best parameter values are tabulated below :

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Bin size list	[6, 15, 22, 23]	Bin size list	[8, 15, 23, 25]	Bin size list	[8, 9, 24, 21]
No. of episodes	10000	No. of episodes	10000	No. of episodes	10000
No. of experiments	5	No. of experiments	5	No. of experiments	5
Learning rate	0.229	Learning rate	0.248	Learning rate	0.100
Discount factor	0.99	Discount factor	0.99	Discount factor	0.99
Policy	'epsilon_greedy'	Policy	'epsilon_greedy'	Policy	'epsilon_greedy'
τ	0.167	τ	0.457	τ	0.199
Decay rate	0.983	Decay rate	0.988	Decay rate	0.999
τ_final	0.098	τ_final	0.082	τ_final	0.065

Table 6: Hyperparameter Values

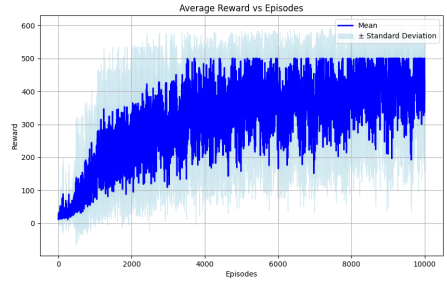


(a) Total Average Reward

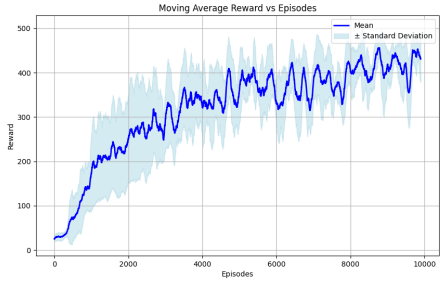


(b) Rolling mean reward

Figure 6: Learning Rate = 0.229 , Mean Regret = 189.62

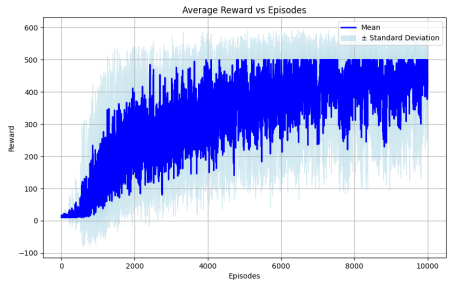


(a) Total Average Reward

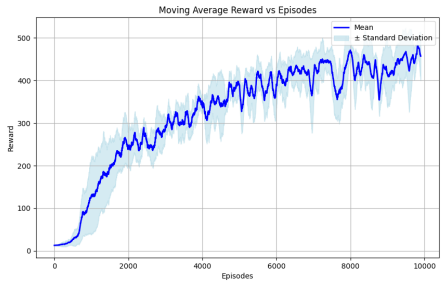


(b) Rolling mean reward

Figure 7: Learning Rate = 0.248 , Mean Regret = 183.19

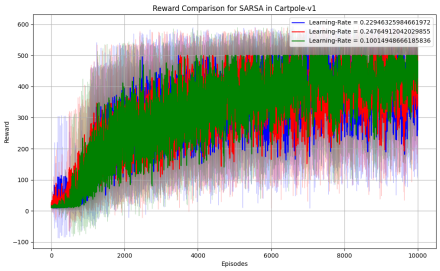


(a) Total Average Reward

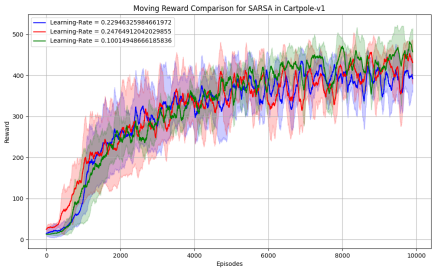


(b) Rolling mean reward

Figure 8: Learning Rate = 0.100 , Mean Regret = 172.62

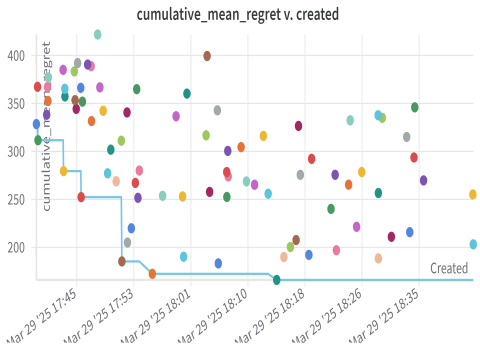


(a) Total Average Reward

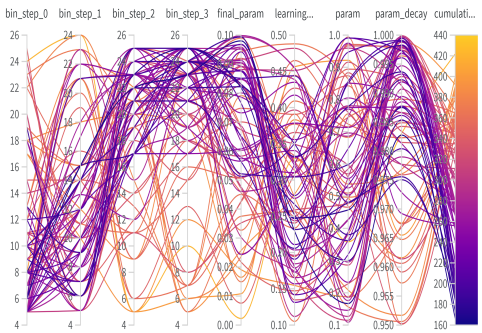


(b) Rolling mean reward

Figure 9: Comparison Plot



(a) Total Average Reward

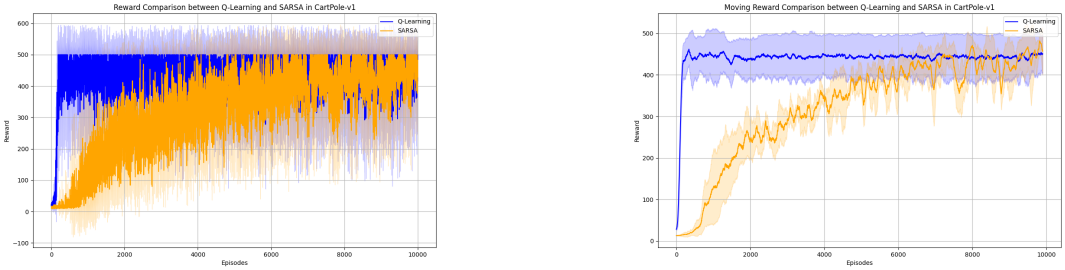


(b) Rolling mean reward

Figure 10: Wandb Plots - CartPole SARSA

5.2 CartPole-v1 : Q-Learning vs SARSA

The plot shows the comparison between the best hyperparameters for the Q-Learning and SARSA algorithms respectively.



(a) Total Average Reward

(b) Rolling mean reward

Figure 11: Q-Learning vs SARSA

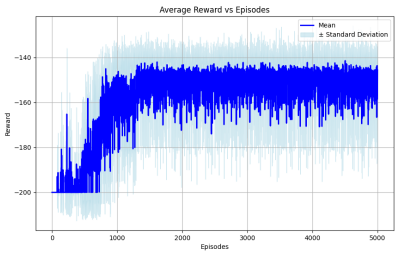
5.3 MountainCar-v0

5.3.1 Q-Learning with Softmax Policy

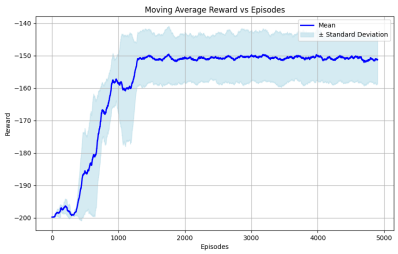
The 3 best parameter values are tabulated below :

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Bin size list	[6, 18]	Bin size list	[6, 16]	Bin size list	[7, 13]
No. of episodes	5000	No. of episodes	5000	No. of episodes	5000
No. of experiments	5	No. of experiments	5	No. of experiments	5
Learning rate	0.331	Learning rate	0.224	Learning rate	0.355
Discount factor	0.99	Discount factor	0.99	Discount factor	0.99
Policy	'softmax'	Policy	'softmax'	Policy	'softmax'
τ	6.345	τ	2.000	τ	1.271
Decay rate	0.942	Decay rate	0.981	Decay rate	0.991
τ_final	0.014	τ_final	0.003	τ_final	0.007

Table 7: Hyperparameter Values

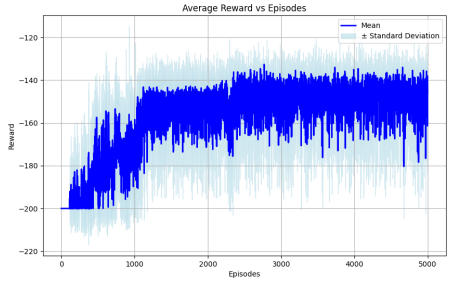


(a) Total Average Reward

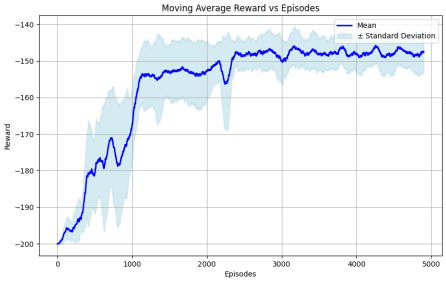


(b) Rolling mean reward

Figure 12: Learning Rate = 0.331 , Mean Regret = 158.27

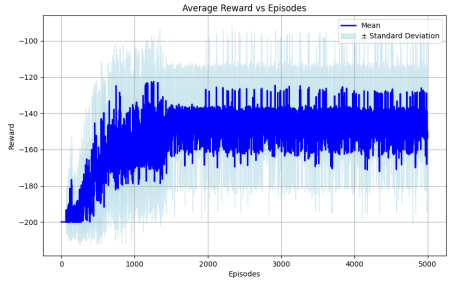


(a) Total Average Reward

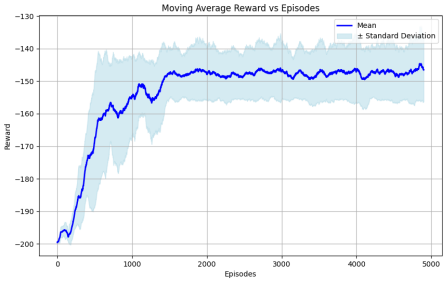


(b) Rolling mean reward

Figure 13: Learning Rate = 0.224 , Mean Regret = 157.06

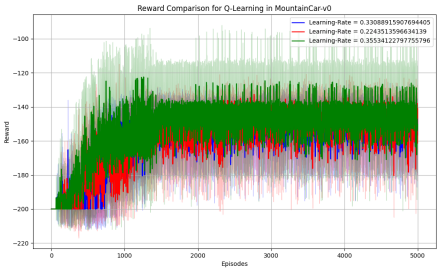


(a) Total Average Reward

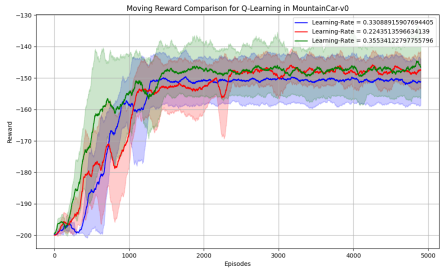


(b) Rolling mean reward

Figure 14: Learning Rate = 0.355 , Mean Regret = 153.74

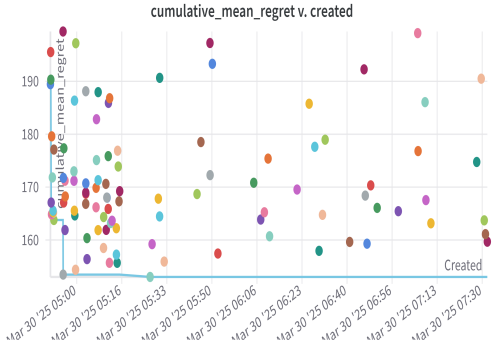


(a) Total Average Reward

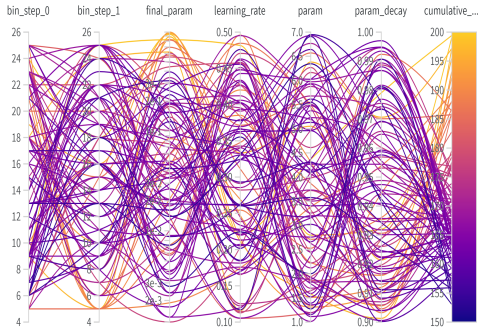


(b) Rolling mean reward

Figure 15: Comparison Plot



(a) Total Average Reward



(b) Rolling mean reward

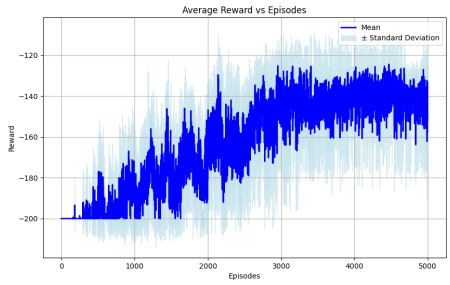
Figure 16: Wandb Plots - MountainCar Q-Learning

5.3.2 SARSA with ϵ-greedy Policy

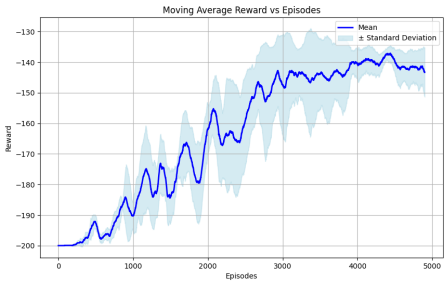
The 3 best parameter values are tabulated below :

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Bin size list	[16, 17]	Bin size list	[19, 12]	Bin size list	[10, 10]
No. of episodes	5000	No. of episodes	5000	No. of episodes	5000
No. of experiments	5	No. of experiments	5	No. of experiments	5
Learning rate	0.154	Learning rate	0.163	Learning rate	0.165
Discount factor	0.99	Discount factor	0.99	Discount factor	0.99
Policy	'epsilon_greedy'	Policy	'epsilon_greedy'	Policy	'epsilon_greedy'
τ	0.749	τ	0.972	τ	0.810
Decay rate	0.955	Decay rate	0.930	Decay rate	0.921
τ_final	0.001	τ_final	0.001	τ_final	0.001

Table 8: Hyperparameter Values

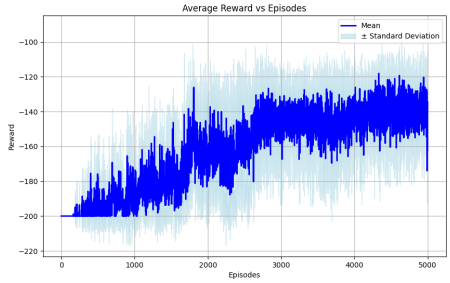


(a) Total Average Reward

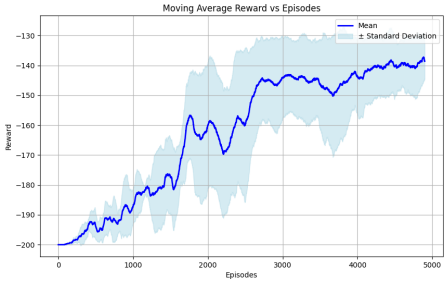


(b) Rolling mean reward

Figure 17: Learning Rate = 0.154 , Mean Regret = 163.12

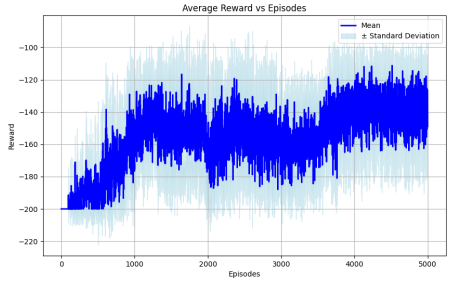


(a) Total Average Reward

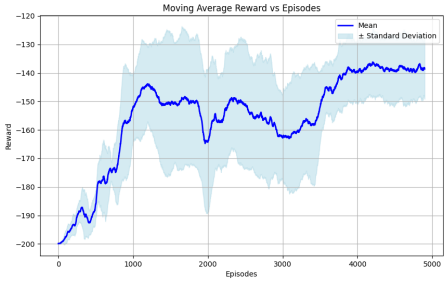


(b) Rolling mean reward

Figure 18: Learning Rate = 0.163 , Mean Regret = 162.47

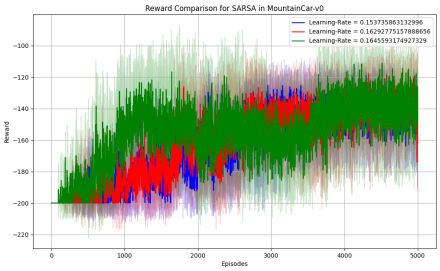


(a) Total Average Reward

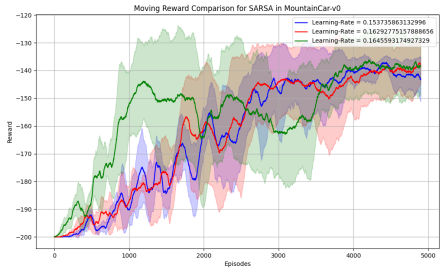


(b) Rolling mean reward

Figure 19: Learning Rate = 0.165 , Mean Regret = 155.85



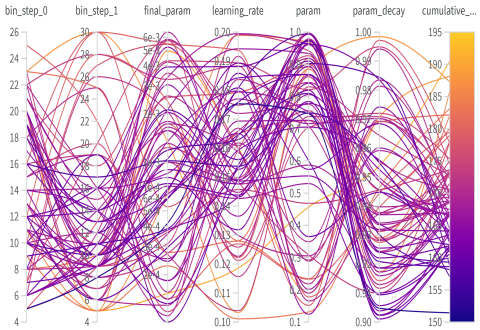
(a) Total Average Reward



(b) Rolling mean reward

Figure 20: Comparison Plot

(a) Total Average Reward

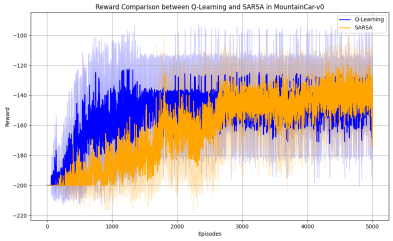


(b) Rolling mean reward

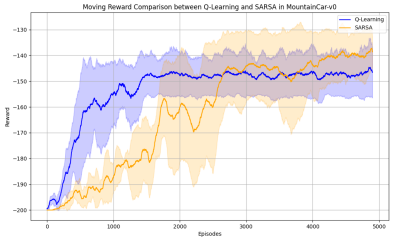
Figure 21: Wandb Plots - MountainCar SARSA

5.4 MountainCar-v0 : Q-Learning vs SARSA

The plot shows the comparison between the best hyperparameters for the Q-Learning and SARSA algorithms respectively.



(a) Total Average Reward



(b) Rolling mean reward

Figure 22: Q-Learning vs SARSA

6 Link to Github

The code along with the README file, requirements.txt file and all the other essential files have been uploaded in this Github repo
The link to the Github classroom repo is attached here.

7 References

- Gymnasium Documentation
- Wandb Documentation

8 Acknowledgements

We would like to thank Prof. Balaram Ravindran and the Teaching Assistants for clearing our queries during the assignment.