Reinforcement Learning Project Report

Nawar Abou Trabi

July 31, 2025

1 The Learning Algorithm

To solve the problem of navigating the grid-world, while minimising the treading on unsafe zones and maximising reward, the *SARSA* algorithms has been implemented, alongside both the $\epsilon - Greedy$ and *Softmax* behavioural policies. The focus of the report is to evaluate the algorithm under different behavioural policies and hyperparameters, in order to optimise the agent's learning and performance in exploiting the higher reward location on the grid.

Why SARSA?

To justify the choice of the algorithm, the two popular choices of $Q - Learning$ and $SARSA$ have been implemented and tested with basic parameters.

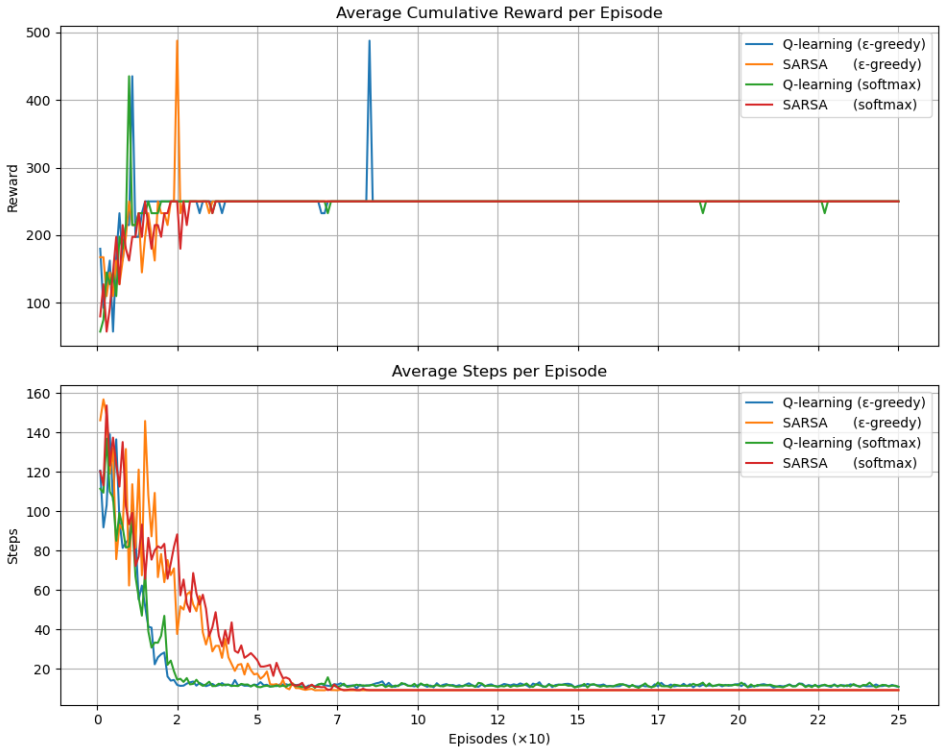

Figure 1: Average learning curve of all learning algorithms with all behavioural policies.

```
final = compare_learning_curves(
    env_class=lambda: GridWorld(
        goal_locations,
        goal_rewards
),
    n_episodes=250,
    n_runs=20,
    epsilon=0.1,
    gamma=0.999,
    temperature=1.0,
    return_final_steps=True
)
```

Figure 2: Parameters of the simulation in fig 1.

```
Q-learning (ε-greedy): 11.6
SARSA      (ε-greedy): 9.0
Q-learning (softmax): 11.55
SARSA      (softmax): 9.0
```

Figure 3: Final average number of steps taken to get to the 250-point reward after 250 episodes for each algorithm/policy combination in fig 1.

As can be seen in figs 1 & 3, although the $SARSA$ algorithm requires more training episodes to reach convergence, it performs better than the $Q-Learning$ algorithm, with either behavioural policy, in terms of number of the final number of steps taken to reach the 250 reward at convergence.

As a result of this quick practical test ran on this specific environment, the $SARSA$ algorithm is the learning algorithm of choice for the agent within this project.

SARSA Algorithm & Derivation

Starting from the Bellman expectation equation for policy π:

$$q^{\pi}(s,a) \;=\; \mathbb{E}_{\pi}\big[R_{t+1} + \gamma\, q^{\pi}(s',a') \mid S_t = s,\; A_t = a\big].$$

Rewriting yields the zero-mean Bellman error:

$$\mathbb{E}_{\pi}\big[R_{t+1} + \gamma\, q^{\pi}(s',a') - q^{\pi}(s,a)\big] = 0.$$

Since q^{π} is unknown, we introduce the estimate $Q(s,a)$. We form a sample-based squared loss over N transitions $(s, a, r^{(i)}, s'^{(i)}, a'^{(i)})$:

$$\mathcal{L}(s,a) = \frac{1}{2N}\sum_{i=1}^{N}\big(Q(s,a) - [\,r^{(i)} + \gamma\, Q(s'^{(i)}, a'^{(i)})]\big)^2.$$

Taking the gradient w.r.t. $Q(s,a)$ and using $N = 1$ (online update) gives the temporal-difference error

$$\delta = r + \gamma\, Q(s',a') - Q(s,a),$$

and the stochastic gradient step

$$Q(s,a) \;\leftarrow\; Q(s,a) + \eta\,\delta.$$

This yields the familiar SARSA update rule:

$$Q(s,a) \;\leftarrow\; Q(s,a) + \eta\big[r + \gamma\, Q(s',a') - Q(s,a)\big].$$

$SARSA$ is thus derived by viewing the Bellman equation as a zero-mean condition, casting it into a loss, and performing gradient descent on observed transitions. This is the foundation for the learning algorithm implemented in the code that is used for running the simulations seen below.

2 Performance with Initial Hyperparameter Values

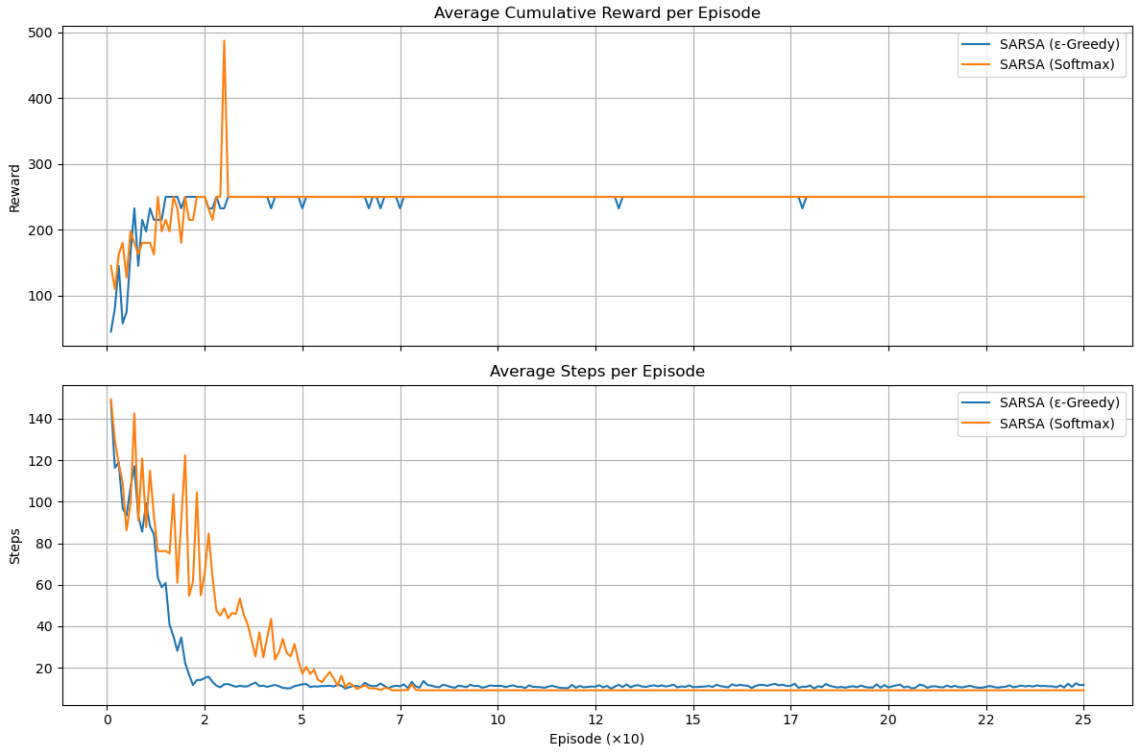

Figure 4: Average learning curve of $SARSA$ with ϵ-Greedy and $Softmax$ behavioural policies.

```
final = compare_learning_curves(
    env_class=lambda: GridWorld(
        goal_locations,
        goal_rewards
),
    methods=['sarsa'],
    policy_types=['epsilon_greedy',
                  'softmax'],
    n_episodes=250,
    n_runs=20,
    epsilon=0.1,
    gamma=0.999,
    temperature=1.0,
    return_final_steps=True
)
```

Figure 5: Parameters of the simulation in fig 4.

```
SARSA (ε-Greedy): 10
SARSA (Softmax): 9
```

Figure 6: Final average number of steps taken to get to the 250-point reward after 250 episodes for each $SARSA$/policy combination in fig 4.

Preferred Movement Policy

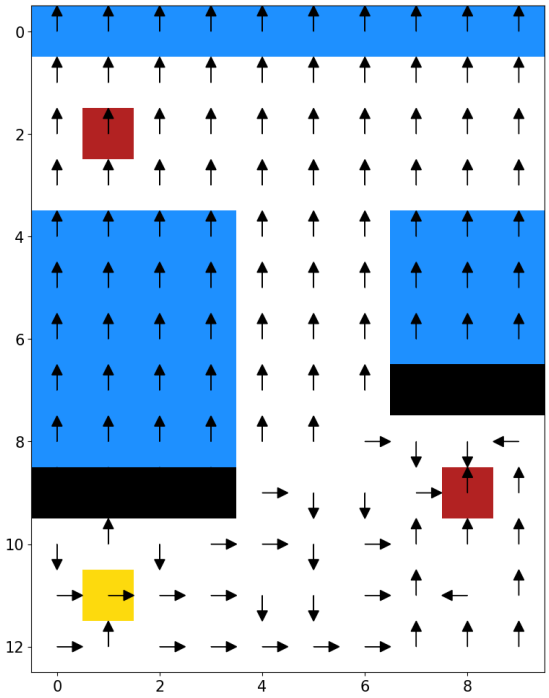

Figure 7: Preferred movement policy of the $SARSA$ algorithm with ϵ-Greedy.

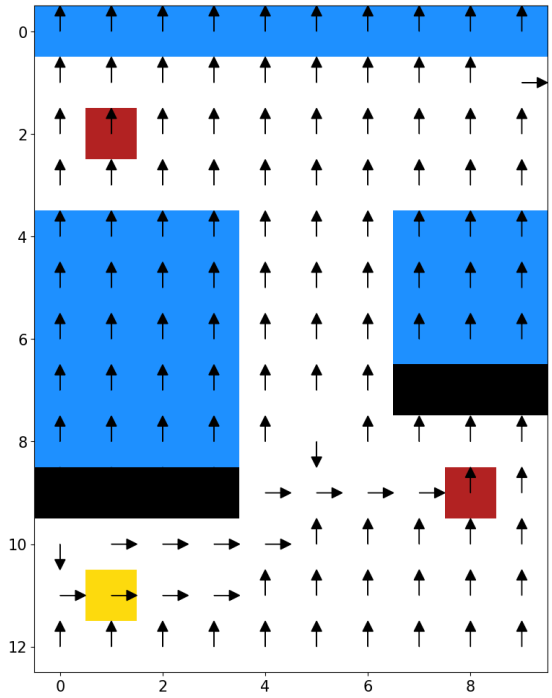

Figure 8: Preferred movement policy of the $SARSA$ algorithm with $Softmax$

Error Deviations

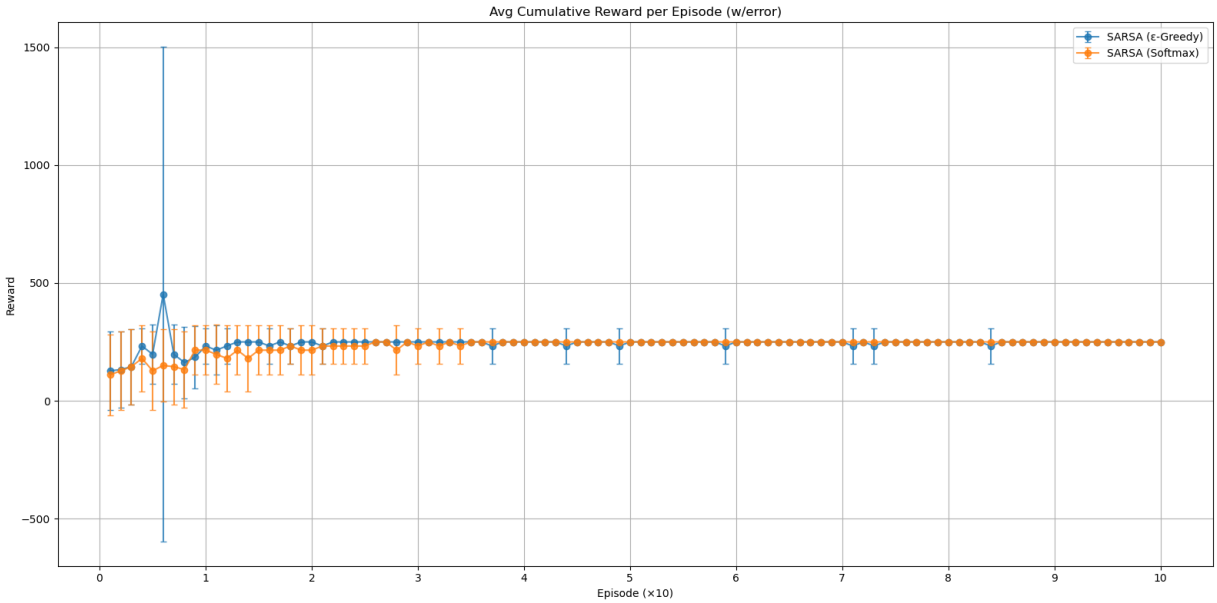

Figure 9: Average reward curve of $SARSA$ with ϵ-Greedy and $Softmax$ behavioural policies.

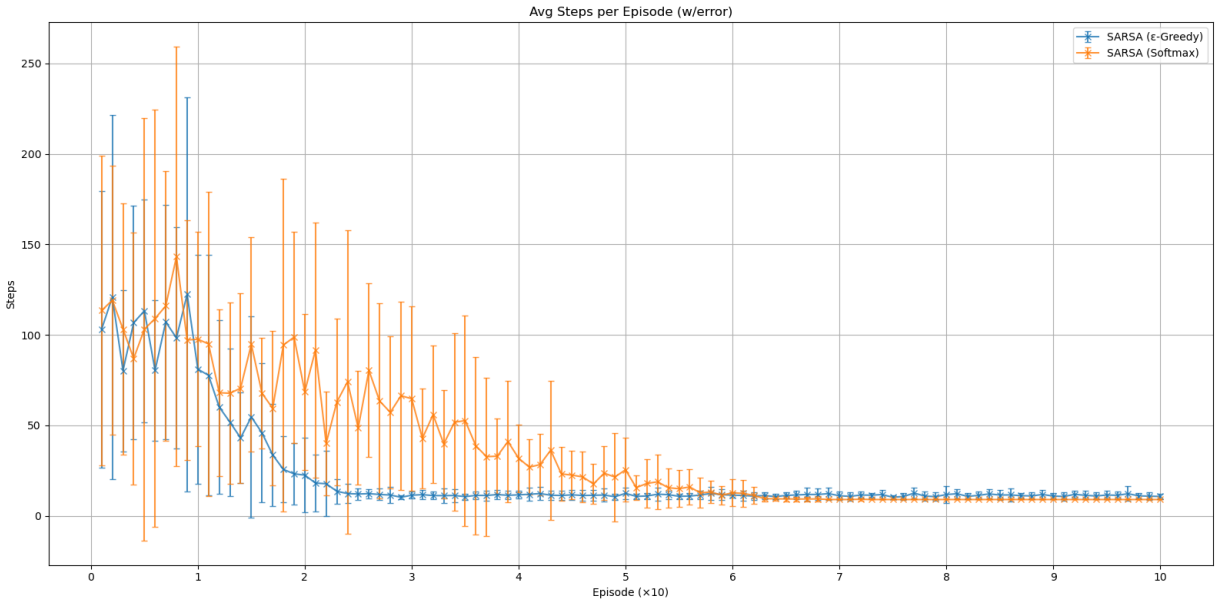

Figure 10: Average steps curve of $SARSA$ with ϵ-Greedy and $Softmax$ behavioural policies.

3 Optimisation of Hyperparameters

The hyperparameters for both the $SARSA/\epsilon$-Greedy and $SARSA/Softmax$ combinations was executed using a grid search written in the ***optimise_parameters()*** function. The grid search works as follows:

1. A range of values is given for each hyperparameter into the ***optimise_parameters()***
2. Starting with the first hyperparameter, and keeping everything else constant, run the simulation for each value within the defined list of values, and store the final average number of steps to reach the reward state.
3. The best performing value (one which results in the minimum number of steps to reach the reward) is stored and used in the next hyperparameter optimisation iteration.
4. Repeat for all the desired hyperparameters.

The initial values of the hyperparameters in the optimisation function are the same as the values used in *Section* 1.

Optimised Hyperparameter Values: $SARSA/\epsilon$-Greedy

```
params, final = optimise_parameters(
    env_class=lambda: GridWorld(
        goal_locations,
        goal_rewards
),
    method='sarsa',
    policy_type='epsilon_greedy',
    param_grid={
        'epsilon': [0.001, 0.005, 0.01, 0.1, 0.2, 0.3, 0.5, 0.9, 0.999],
        'gamma':   [0.001, 0.01, 0.5, 0.9, 0.95, 0.999],
        'eta':     [0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.9]
    },
    n_episodes=100,
    n_runs=20
)
```

Figure 11: Parameters of the grid search.

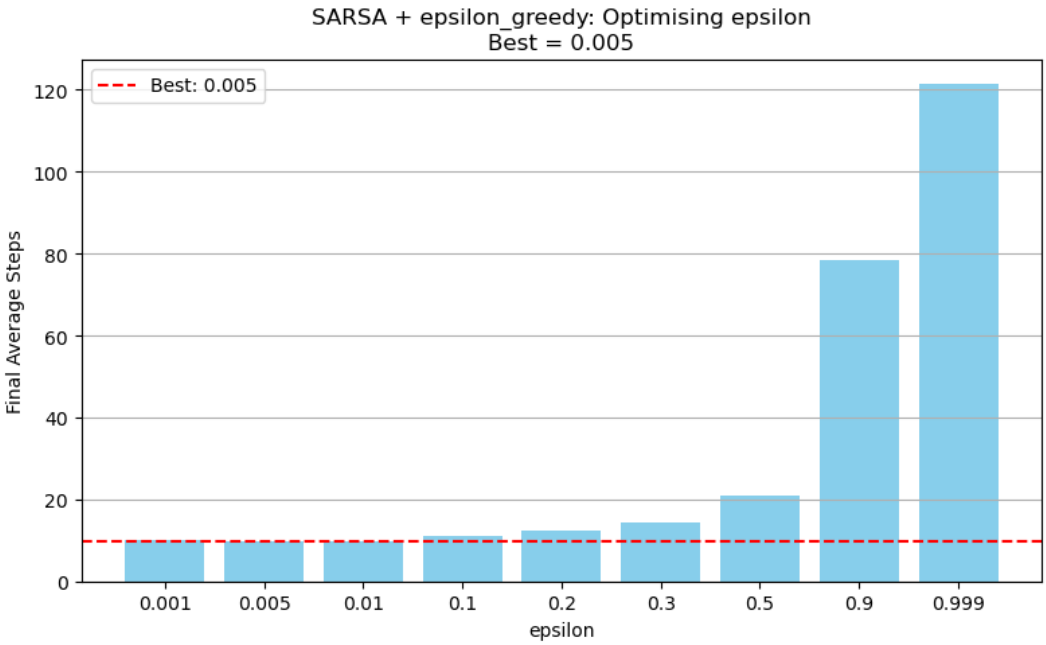

Figure 12: Distribution of the average final number of steps amongst each ϵ value.

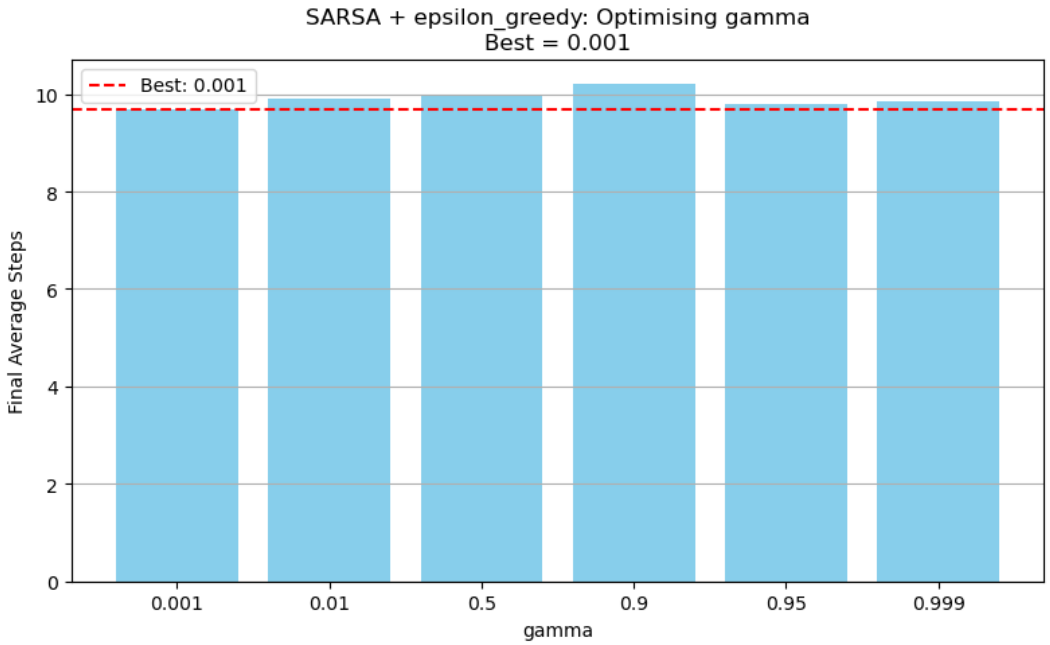

Figure 13: Distribution of the average final number of steps amongst each γ value.

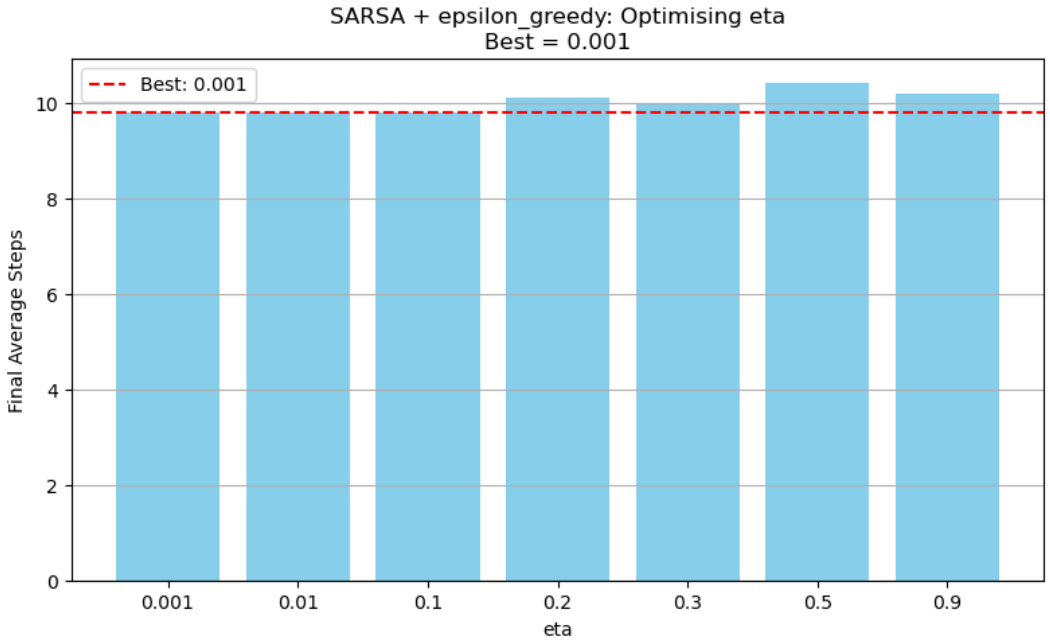

Figure 14: Distribution of the average final number of steps amongst each η value.

Giving optimised parameter values of:

- ϵ: 0.005
- γ: 0.001
- η: 0.001

And a minimum average number of steps value of **9.8** to reach the 250 reward.

Optimised Hyperparameter Values: $SARSA/Softmax$

```
params, final = optimise_parameters(
    env_class=lambda: GridWorld(
        goal_locations,
        goal_rewards
),
    method='sarsa',
    policy_type='softmax',
    param_grid={
        'temperature': [0.1, 0.2, 0.5, 0.9, 1, 1.25, 1.5, 1.75, 2],
        'gamma':   [0.001, 0.01, 0.5, 0.9, 0.95, 0.999],
        'eta':     [0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.9]
    },
    n_episodes=100,
    n_runs=20
)
```

Figure 15: Parameters of the grid search.

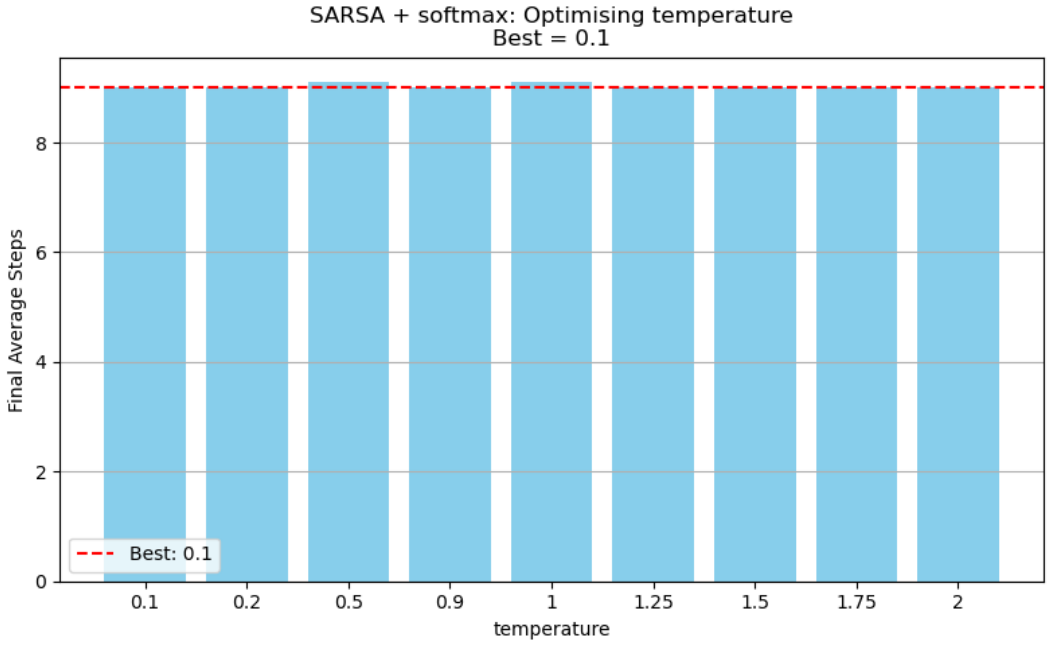

Figure 16: Distribution of the average final number of steps amongst each τ value.

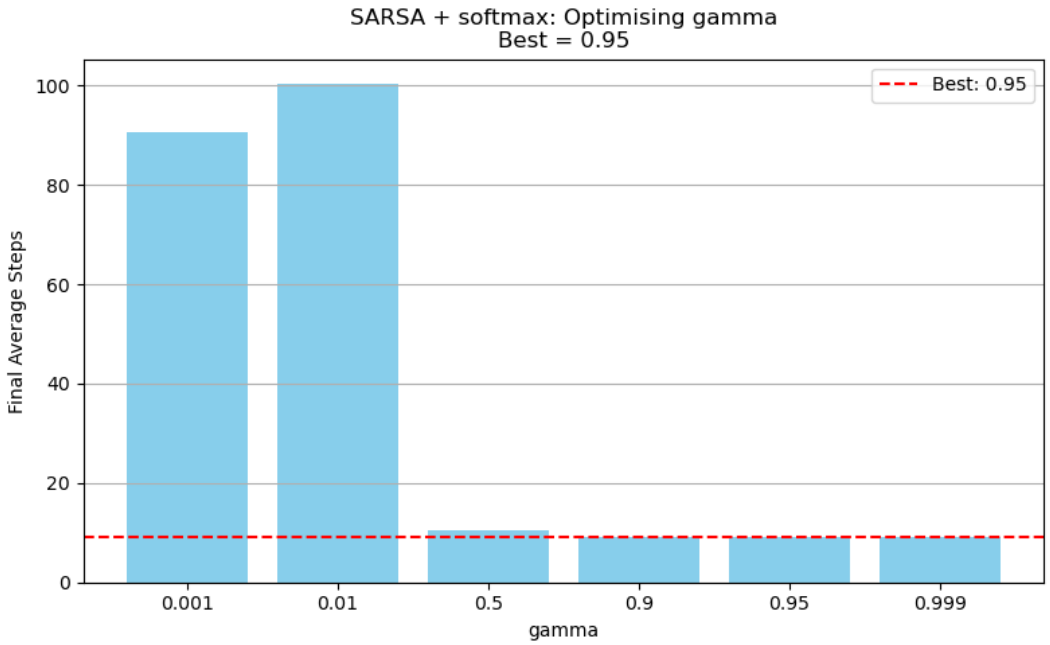

Figure 17: Distribution of the average final number of steps amongst each γ value.

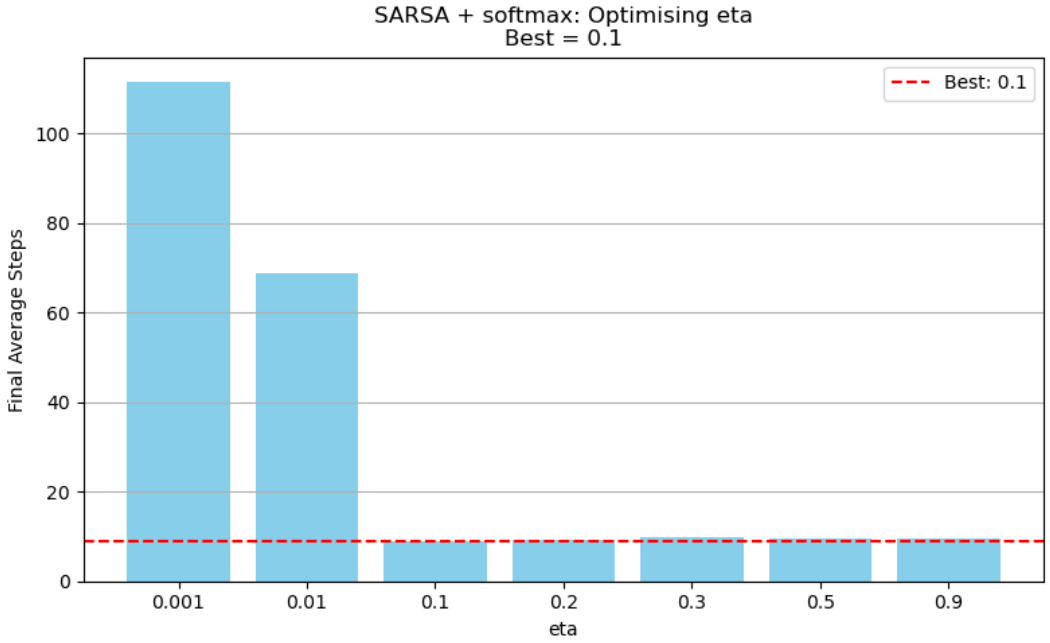

Figure 18: Distribution of the average final number of steps amongst each η value.

Giving optimised parameter values of:

- τ: 0.1
- γ: 0.001
- η: 0.001

And a minimum average number of steps value of **9.0** to reach the 250 reward.

It is an interesting observation to see that both the ϵ-Greedy and $Softmax$ behavioural policies prefer lower hyperparameter values (of 0.005 and 0.1 respectively), as well as all the hyperparameters of the $SARSA$ algorithm. As it provides a lower average number of steps than the $Q-Learning$ algorithm, only the $SARSA/Softmax$ combination will be used hereafter.

Summary of Hyperparameter Effects:

Parameter	Too Small	Too Large
η	Slow learning	Oscillation
γ	Myopic policy	Slow convergence
ϵ	Stuck in local	Noisy behaviour
τ	Greedy (no E)	Random (no X)

E: exploration, **X**: exploitation.

It is important to note that minute changes to the hyper-parameters have little to no effect on the performance of the agent. Taking full advantage of this, parameter values that looked neater were preferred in optimisation (e.g. 0.999 over 0.997 or 0.991).

4 Challenges of a Second Reward Location

The distant 5000-point artefact introduces two interrelated difficulties. First, the agent naturally encounters the nearby 250-point reward far more often, since it lies along shorter trajectories. Second, with a standard discount factor, even successful visits to the 5000-point artefact bring about a discounted return that may fall below the immediate 250-point payoff. As a result, the learned policy consistently prefers the smaller, closer reward.

A practical solution is *optimistic initialisation*. Because the current SARSA initialises all Q-values to zero, any positive reward appears advantageous. By instead setting every Q-value to a large constant — of 5,000 for instance — the agent treats the 250-point outcome as a much more significant reduction from its initial expectation of 5,000, and continues to explore. This optimism effectively drives the agent to traverse the longer route and secure the higher-valued distant reward over the nearer, lower reward.

Performance with Optimised Hyperparameters + 'Optimism'

The hyperparameter optimisation process has been executed for the simulations ran with optimism, but has not been included in this section for simplicity. The optimised hyperparameter values for the $SARSA/Softmax$ combination are shown in figure 20.

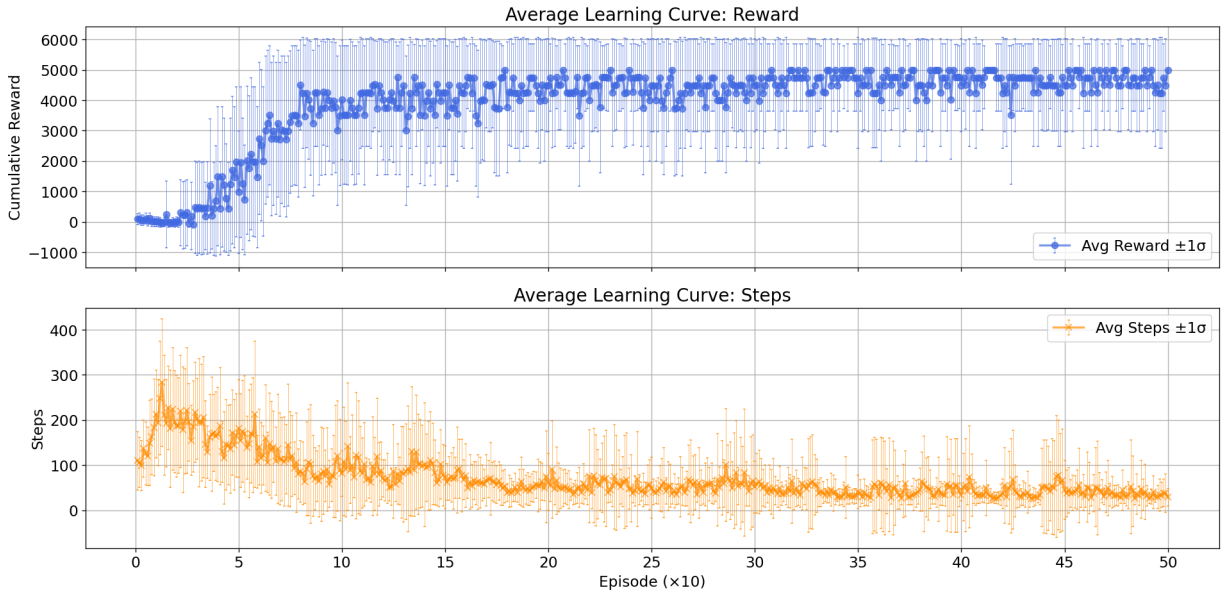

Figure 19: Average learning curve of $SARSA$ with $Softmax$ behavioural policy.

```
final = plot_avg_learning_curves(
    lambda: GridWorld(goal_locations, goal_rewards),
    n_episodes=500,
    n_runs=20,
    method='sarsa',
    policy_type='softmax',
    eta=0.1,
    temperature=0.1,
    gamma=0.999,
    optimistic=True,

)
```

Figure 20: Parameters of the simulation in fig 19.

Figure 21: Final average number of steps taken to get to the 5000-point reward after 5000 episodes for the $SARSA/Softmax$ combination.

A value of 500 episodes was used to fit figure 19 on the screen. Final testing used 5000 episodes for training.

Preferred Movement Policy

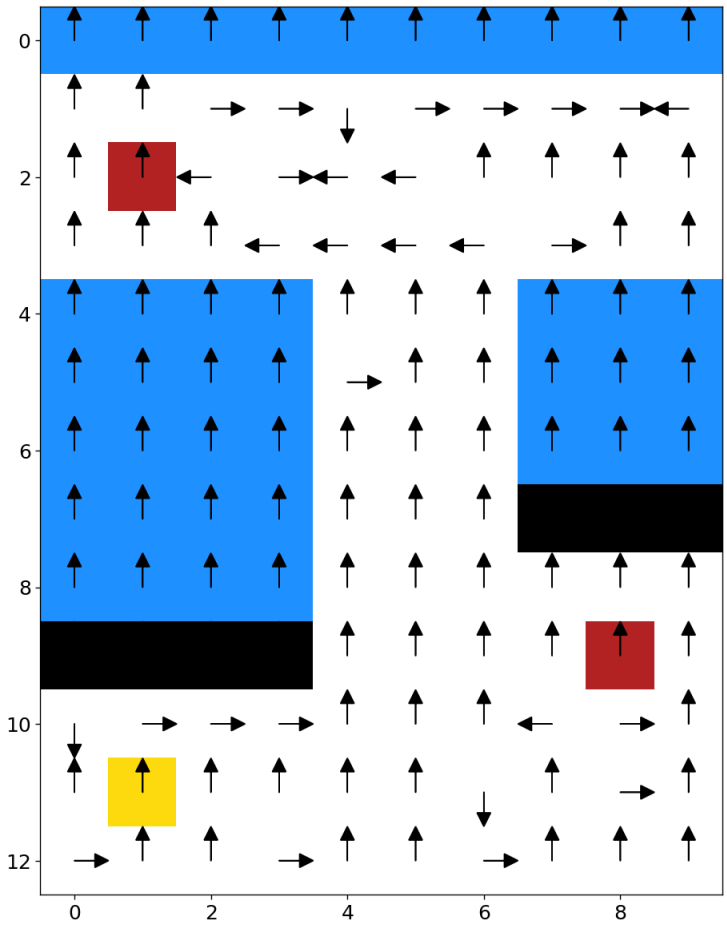

Figure 22: Preferred movement policy of the $SARSA$ algorithm with $Softmax$, taking **17** steps to reach the 5000 reward (after training for 5000 episodes).

5 Final Remarks: The Breadwinner

The optimistic $SARSA$ learning algorithm with the $Softmax$ behavioural policy and the following parameters: $\tau = 0.1$, $\eta = 0.1$ and $\gamma = 0.999$ performed the best, reaching the 5000 reward in **17** steps (and 20.45 on average).

When going for a reward situated at a further distance, it has been shown that higher γ values are crucial. This is because a higher γ value emphasises long term rewards as shown in *Section* 3's hyperparameter summary table.

GitHub Link

`https://github.com/NawarOfThings/project_ReinforcementLearning`