Comparing Q-learning and SARSA in a Grid World Problem

1st Simon du Toit
Applied Mathematics Department
Stellenbosch University

***Abstract*—In this paper I compare two reinforcement learning algorithms: SARSA and Q-learning. Q-learning is a popular off-policy reinforcement learning algorithm, while SARSA is an on-policy alternative. The algorithms are applied to a simple grid world problem. I find that Q-learning converges more rapidly than SARSA, but both algorithms converge to equally optimal policies. This implies that Q-learning may be the more efficient algorithm for simple environments.**

I. Introduction

I investigate the performance of two reinforcement learning algorithms, SARSA and Q-learning, on a simple grid world problem. First I explain the theoretical background for each algorithm. The grid world environment I use to evaluate the algorithms is then described in detail. I explain my implementation of the algorithms and how I tune their hyperparameters. Finally, I present and discuss the results of applying the algorithms to find an optimal policy for the environment.

II. Background

In this section I explain theory around the Q-learning and SARSA algorithms.

A. SARSA

SARSA is a reinforcement learning algorithm which is used to estimate the optimal policy for an agent in a given environment[1]. It does so by having the agent explore different states in the environment and building an estimate of the Q-values, which are the values of each action from each state.

In SARSA the agent selects actions ϵ-greedily according to the current estimate of the Q-values. On each iteration the Q-value for the current state and action is updated according to the TD(0) (temporal difference) algorithm. SARSA is an on-policy algorithm, which means that the same policy is used to control the agent and update the Q-values. Pseudocode for the SARSA algorithm is presented in Algorithm 1.

The ϵ is annealed over the course of training to shift the agent from exploration to exploitation. The two other hyperparameters are α (learning rate), which determines how much Q-values are updated at each iteration, and γ (discount), which weighs the influence of future rewards relative to the immediate reward.

B. Q-learning

Similar to SARSA, Q-learning is a reinforcement learning algorithm which estimates the Q-values of actions at states in the environment[1]. However, it differs from SARSA in that it is an off-policy algorithm. At each iteration the next action is chosen greedily, but this action is only used to update the Q-value and not to control the agent in the next iteration. Pseudocode for the Q-learning algorithm is presented in Algorithm 2.

Algorithm 1 Pseudocode for one episode of the SARSA algorithm. S, S', A, A' refer to the current and future state, and current and future action respectively. $\mathbf{Q}$ is a representation of the Q-values and R is the current reward. $epsilon$, α and γ are hyperparameters.

```
Initialise S
Choose A from S ε-greedily with respect to Q.
while S not terminal state do
    Get R and S' from A
    Choose A' from S' ε-greedily with respect to Q.
    Q(S,A) ← Q(S,A) + α(R + γQ(S',A') − Q(A,S))
    S ← S'
    A ← A'
end while
```

Algorithm 2 Pseudocode for one episode of the Q-learning algorithm. S, S', A, A' refer to the current and future state, and current and future action respectively. $\mathbf{Q}$ is a representation of the Q-values and R is the current reward. $epsilon$, α and γ are hyperparameters.

```
Initialise S
while S not terminal state do
    Choose A from S ε-greedily with respect to Q.
    Get R and S' from A
    Choose A' from S' greedily with respect to Q.
    Q(S,A) ← Q(S,A) + α(R + γQ(S',A') − Q(A,S))
    S ← S'
end while
```

III. Problem

The problem is for the agent to learn the task of navigating a simple 8×8 grid world environment. The grid world consists of empty space surrounded by an impassible wall. The agent is initialised in the top left and must reach the bottom right open space to receive a reward. The agent loses reward whenever it moves into a wall and receives a greater reward the less actions it uses to reach the goal. The state is based on the agent's

limited vision in front of it and the action space consists of {turn left, turn right, move forward}.

IV. Implementation

Both algorithms are programmed in Python, using the `numpy` library. The environment is simulated using `gymnasium`.

Since both the state space and the action space are discreet, the Q-values are stored in a lookup table. This lookup table is implemented as a dictionary with an array of Q-values for each action at each index. The state is expressed as an integer matrix representing the view of spaces in front of the agent. A hash value is derived from a state by converting it to a byte-string. This hash value is used to index into the Q-table.

The agent is trained for 3000 episodes. The value of ϵ is initialised at a maximum value and annealed towards a minimum value by multiplying by 0.999 after each episode. The hyperparameters: minimum and maximum ϵ, α and γ are tuned using 30 iterations of random search. For minimum and maximum ϵ I search over $(0, 0.5)$ and $(0.5, 1)$ respectively and for α and γ I search over $(0, 1)$. The parameter configuration which results in the highest average reward over the course of training is chosen.

V. Results

In this section I present the results of my experiments in tuning and comparing the SARSA and Q-world algorithms on the described grid world problem.

A. Hyperparameter Tuning and Training

Model	Max ϵ	Min ϵ	α	γ
SARSA	0.61	0.03	0.32	0.44
Q-learning	0.56	0.12	0.89	0.67

Fig. 1. Best hyperparameter values found by random search for each algorithm.

The optimal hyper parameter configurations for each algorithm are given in Figure 1. The maximum ϵ value is quite low in both cases, indicating that for such a simple problem the algorithms are able to focus on exploitation early in training.

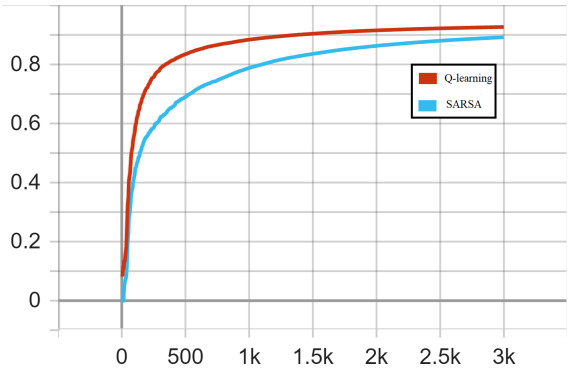

Fig. 2. Average reward (y-axis) over training iterations (x-axis).

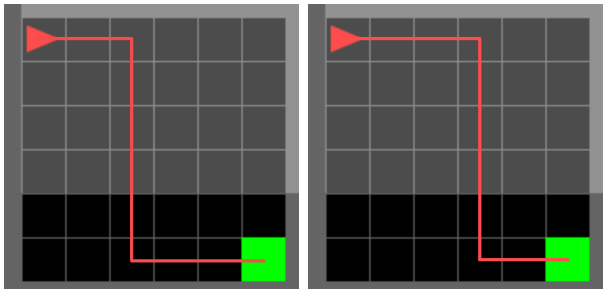

Fig. 3. Optimal policy learned by SARSA (left) and Q-learning (right).

There is a large difference in the optimal α values. The tuned Q-learning algorithm updates its Q-value estimates much faster than the tuned SARSA algorithm. The difference in γ values is small, with Q-learning weighing future reward a bit more than SARSA does.

Both algorithms are trained using these optimal parameters. The average reward over the course of training is shown in Figure 2. From this it is clear that the tuned Q-learning implementation converges to the optimal policy more rapidly than SARSA. This could be due to the increased greediness in the Q-learning algorithm, or the higher value of α compared to SARSA.

B. Optimal Policy

Model	Completion rate	Averate steps	Average reward
SARSA	1.0	12	0.9578
Q-learning	1.0	12	0.9578

Fig. 4. Results of using optimal policy learned by each algorithm

The optimal policies found by SARSA and Q-learning are shown side-by-side in Figure 3. The algorithms learn very similar policies.

Each of these policies is used to control the agent for 1000 iterations. The average reward, average number of actions and completion rate is recorded and presented in Figure 4. The optimal policies learned by each algorithm are equally optimal. This indicates that both algorithms are able to find a good policy for this problem. It may also be due to the large number of equivalently optimal policies that can be used in this environment..

VI. Conclusion

I have implemented the SARSA and Q-learning algorithms and used them to find an optimal policy for a simple grid world problem. Both algorithms find equally optimal policies, but Q-learning converges to its policy much faster than SARSA does. This shows that Q-learning is the more efficient algorithm for this problem and perhaps simple environments in general.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction Second Edition*. Cambridge: MIT Press, 2018.