Reinforcement Learning Threshold Assignment

220152703

1 The SARSA Algorithm

We can model the robot's decision-making process with a Markov Decision Process (MDP), with:

- A set of states S representing the possible positions the robot can have.
- A set of actions A available to the robot.
- A transition probability function $P(s' \mid s, a)$ that defines the likelihood of moving to state s' after taking action a in state s.
- A reward function $R(s, a, s')$ specifying the immediate reward received when taking action a in state s and transitioning to s'.

The robot starts from a state s with the goal of selecting actions that maximise the expected return. The agent selects actions according to a policy $\pi(a \mid s)$.

The total return the agent receives is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

$$G_{t+1} = R_{t+2} + \gamma R_{t+3} + \dots$$
$$\Rightarrow G_t = R_{t+1} + \gamma G_{t+1}$$

We denote the expected return of taking an action a in state s as $q^\pi(s, a)$ defined as:

$$q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

This is equivalent to:

$$q^\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q^\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

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

To maximise expected return in any state, the robot should choose the action that maximises $q^\pi(s, a)$. Since the true values of $q^\pi(s, a)$ are unknown, we introduce a parameterised function $Q(s, a)$ to estimate them. The expectation then becomes approximately zero:

$$\mathbb{E}_\pi[R_{t+1} + \gamma Q(s', a') - Q(s, a) \mid S_t = s, A_t = a] \approx 0$$

We define a squared loss function around this quantity to make it minimizable:

$$L(s, a) = \frac{1}{2}(Q(s, a) - (r + \gamma Q(s', a')))^2$$

Assuming $Q(s', a')$ is known and fixed, the partial derivative of the loss function with respect to $Q(s, a)$ is:

$$\frac{\partial L(s, a)}{\partial Q(s, a)} = Q(s, a) - (r + \gamma Q(s', a'))$$

To update $Q(s, a)$, we apply the online update rule:

$$Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma Q(s', a') - Q(s, a))$$

Algorithm:

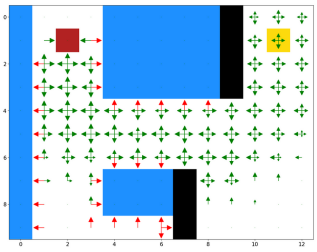
1. Initialise $Q(s, a)$ arbitrarily for all $s \in S$, $a \in A$.
2. For each episode:
   (a) Initialise state s
   (b) Choose action a from s using a policy derived from Q
   (c) For each step of the episode:
      i. Take action a, observe reward r and next state s'
      ii. If s' is terminal:
         - Update $Q(s, a) : Q(s, a) \leftarrow Q(s, a) + \eta(r - Q(s, a))$
         - Break
      iii. Choose action a' from s' using a policy derived from Q
      iv. Update $Q(s, a) : Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma Q(s', a') - Q(s, a))$
      v. Update state and action: $s \leftarrow s', a \leftarrow a'$

2 Training

Two different environments were used for training. The environments we both (9*13) grids, with cliffs of reward -100, and impassable walls. Environment 1 had one reward of 5000 at (1,2). Environment 2 had that reward as well as another at (8, 11) with value 250.

Multiple training runs were completed in order to maximise the agent's expected return, with varying hyperparameters and policies.

Initially, the SERSA algorithm with epsilon-greedy policy was used on the simpler 1 reward environment.



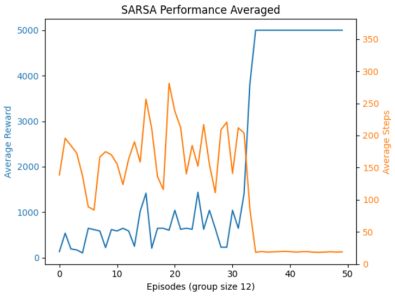


Figure 1: num episodes = 5000, η= 0.5, γ = 1 - 0.01, ϵ = 0.1

With these parameters SERSA very quickly learned the optimal path. γ is set very close to 1 so that the agent cares about actions it takes far into the future (like when it is at the far reward).

The same algorithm and parameters didn't work as well on the 2 reward environment, struggling to explore enough to find the higher reward and getting stuck in a local minima.

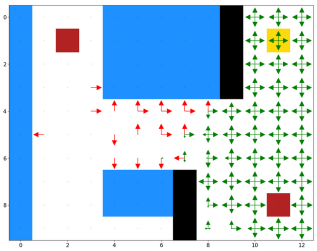
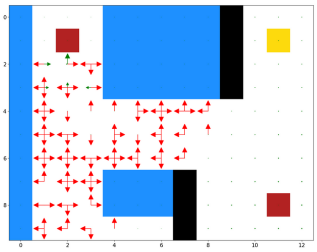


Figure 2: num episodes = 5000, η= 0.5, γ = 1 - 0.01, ϵ = 0.5



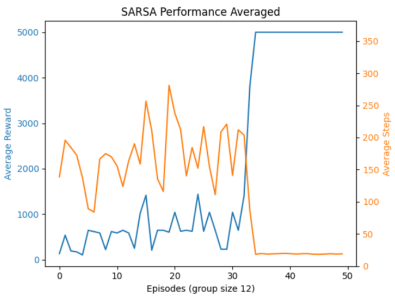
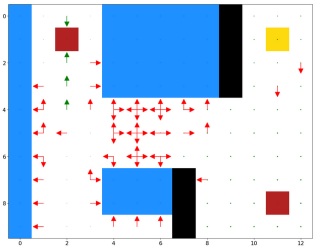


Figure 3: num episodes = 5000, η= 0.5, γ = 1 - 0.01, ϵ = 0.9

Increasing the ϵ value means the agent has a higher chance of exploring and so can get further on its random walk during the episode. This allows it to reach the far reward, but the high ϵ value also means that the agent follows the learned ideal direction less often and therefore will not necessarily make it back to the far reward to propagate the expected reward back towards the start.

This would suggest that an ϵ-schedule may improve the robot's ability to learn Q. Using the formula ϵ = start * (decaytime - step) / decaytime, ϵ will decay over the episode. This should allow the agent to initially explore, and then over time will take greedy actions more and more frequently.



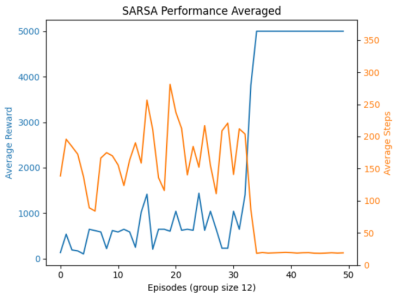
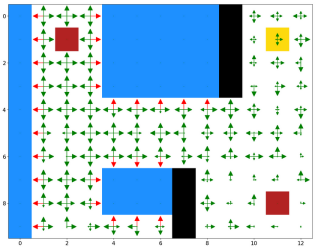


Figure 4: num episodes = 5000, η= 0.98, γ = 1 - 0.01, ϵ_0 = 1, ϵ decay = 65

This ϵ schedule did not improve the ability of the agent. This could be because many of the agents end up taking a random walk off of a cliff, ending the episode early and adding little new information to the system, and adding negative value to the states on the bridge which must be passed to collect the maximum reward.



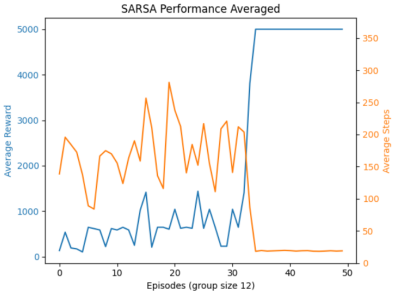
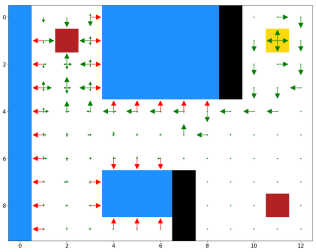


Figure 5: num episodes = 4000, η= 0.98, γ = 1 - 0.0001, ϵ_0 = 1, ϵ decay = 250

This can be solved by amending the epsilon-greedy policy, making sure it only randomly selects an action with negative expected value if there are no positive valued actions. We name this new policy epsilon-greedy-safe.

Now that the passage is safe once the cliffs have been discovered, it is no longer given a low expected return. However, our agent is still not consistently achieving the best reward. This is because during every episode the agent is randomly moving for a lot of its lifespan. This means when it comes time to take its greedy actions, it may be attracted to the wrong reward or it may not make it to a reward before the maximum number of steps is up.



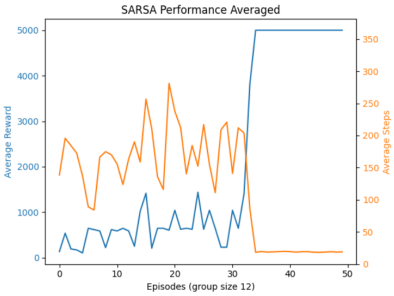


Figure 6: num episodes = 600, η= 0.85, γ = 1 - 0.0001, Stage 2 time = 400, Stage 2 $\epsilon = 0$

In order to fix this, we update the ϵ-schedule to change based on the number of elapsed episodes instead of the step in each episode. We also change it to explore randomly for a fixed number of steps (safely), then act greedily. This means that after a fixed number of episodes, the agent will act according to the learned values and will achieve the maximum reward consistently.

There are some issues with the learned policy. Namely, there are many infrequently visited states which don't get a chance to learn about the newly updated values of its surrounding states. This causes suboptimal behaviour in these states, like cycling or in some cases receiving negative reward. This could potentially be fixed by implementing a second exploration epoch with a small value for ϵ, so that the agent will still go towards the reward but will also explore.

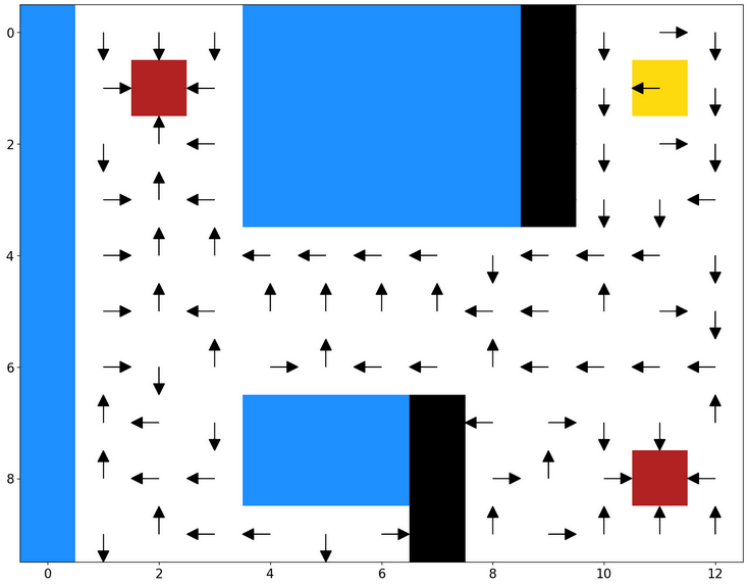


Figure 7: Figure 6's policy (best)

3 Conclusion

In conclusion, the SERSA algorithm with a safe epsilon-greedy policy and ϵ-schedule is adequate to successfully learn the gridworld environment, and to allow the agent to maximise total reward. Policy tuning and hyperparameter scheduling are very powerful tools that allow complicated scenarios to be modelled.