Assignment 2: Model-Free Reinforcement Learning in the ShortCut Environment

Adrien Im (s3984389) & Bence Válint (s3796426)
Group Number: 4

1 INTRODUCTION

Reinforcement Learning (RL) is a framework for making sequential decisions by an agent who learns through interaction with its environment. Within this framework, an important distinction is made between model-based and model-free RL. Model-based approaches are based on a model of the environment's dynamics for planning whereas model-free approaches are operated purely through trial and error, without knowledge of state transitions and reward functions. This allows model-free RL approaches to be useful in environments with complex dynamics that are difficult to model [1]. In this report, we will examine four different model-free RL algorithms: Q-Learning, SARSA, Expected SARSA, and n-step SARSA. These algorithms will be explored in the context of the ShortCut Environment.

1.1 The ShortCut Environment

The ShortCut environment is a 12×12 grid with two blue starting positions, randomly chosen at the start of each episode. Red cells represent "cliffs", which return the agent to the start and cause a penalty of -100. Every other step has a reward of -1. There are four possible actions in the environment: up, down, left, and right. If the agent tries to move outside of the grid, it will stay in place and get a -1 reward. The objective is to reach the single green goal state, which ends the episode. The agent must learn to maximize cumulative rewards by avoiding cliffs, staying within bounds, and finding the shortest safe path to the goal.

2 METHODS

2.1 Q-Learning

The first explored model-free reinforcement learning algorithm will be Q-Learning. Q-Learning maintains a state-action table with an expected reward of each state-action pair and follows a greedy policy to obtain its goal.

To walk through the algorithm the first step is to create Q-table with each possible state-action pair initialized to zero or an arbitrary value. The next step is to update the state-action values for a number of episodes. Next up is to decide the exploration-exploitation policy which in this case will be an epsilon-greedy policy. Starting from one of the starting states using our policy we obtain an action and update the state action value using the formula: $Q(S,A) = Q(S,A) + \alpha[R + \gamma * \max_{A'} Q(S',A') - Q(S,A)]$. In the formula $Q(s,a)$ is the state-action expected reward, α is the learning rate, R is the reward for taking action A at state S, $Q(S',A')$ is the expected future rewards for taking the best action A' from the next state. In the equation the learning rate α is used to determine how much of the previous state-action value gets overwritten, with 1 meaning a complete overwrite and 0 meaning no effect. γ is used to determine the importance of future rewards, with 1 meaning that all long-term rewards are considered and 0 meaning only intermediate rewards are considered. Then the next state is taken as the current state and the state-values are updated until the goal state is reached. For each episode this process is repeated. When this loop is repeated for the number of episodes a state-action table is obtain which can be used along with a greedy policy to find the best path to the goal state.

2.2 SARSA

Another model-free reinforcement learning algorithm is State-Action-Reward-State-Action (SARSA). It also utilizes a state-action table to store the reward of each state-action pair.

Similarly to the Q-Learning algorithm this algorithm also starts off by initializing a state-action table with arbitrary values except the goal state, which is set to 0. In this study this equation utilizes the same exploration-exploitation policy, the epsilon greedy policy. The algorithm start off by looping through each episode and updating the state-action values within each episode. This is done by choosing an action A according to the policy π, the epsilon-greedy policy. Then the action A is taken and reward R and next state S' are observed. The action from the next state A' is also selected by the policy π. Subsequently the following formula is applied to update the state-value of the original state-action pair $Q(S,A)$: $Q(S,A) = Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$.

To break down this equation the same symbols are used as in Q-Learning: α is the learning rate, γ is the future-rewards tradeoff, and $Q(S',A')$ is the future rewards of action A' in state S' according to the policy π. After updating the state-action value the new state is used as the current state and this is repeated until the goal state is reached. When the previously described process is done for the number of episodes, a final table of state-action values are found and like the Q-Learning algorithm this table can be used to find the path to the goals state using a greedy algorithm [1].

2.3 Expected SARSA

Expected SARSA is another model-free algorithm that builds upon the SARSA algorithm by changing the manner in which future rewards are calculated. Instead of selecting the next action, and using the corresponding Q-value, Expected SARSA will calculate an expected value over the totality of possible actions using probabilities given by the policy. As in the previous policies, the Q-table is initialized with arbitrary values and an ϵ-greedy policy is used to trade off exploration and exploitation. During training, at each step of an episode, the agent observes the current state S, selects an action A, observes the reward R, and the next state S'. After that, instead of selecting the next action, the algorithm calculates the expected Q-value using the policy probabilities. The update formula is: $Q(S,A) = Q(S,A) + \alpha\,[R + \gamma \sum_{a'} \pi(a'|S')Q(S',a') - Q(S,A)]$. Where $\pi(a'|S')$ is the probability of taking action a' in state S'. This update is performed until the agent reaches the goal. Then the next episode begins. Once the agent has been trained over all the episodes, the resulting Q-table is used to get the optimal path using a greedy policy [1].

2.4 n-step SARSA

Lastly, n-step SARSA is another extension of the SARSA algorithm that includes n steps in the future for the update of the Q-value. It does not use a single immediate reward, but instead makes use of a cumulative discounted return over n steps. Like the previous methods, the algorithm starts by initializing the Q-table with arbitrary values. During training, the agent stores states, actions, and rewards for n steps. After these n steps, the reward G_t at time t is calculated using: $G_t = \sum_{k=0}^{n-1} \gamma^k R_{t+k} + \gamma^n Q(S_{t+n}, A_{t+n})$. The first part of the equation, $\sum_{k=0}^{n-1} \gamma^k R_{t+k}$ represents the cumulative discounted reward collected over the next n steps. The second part is responsible for the future rewards after the n steps starting from state S_{t+n} and action A_{t+n}. Subsequently, the update rule is applied to update the state-action values: $Q(S_t, A_t) = Q(S_t, A_t) + \alpha \, [G_t - Q(S_t, A_t)]$. Here, α is the learning rate, and γ is the discount factor. After the agent updates the state-action values, the new starting state is $t + 1$, and the process is repeated for the next step in each episode [1].

3 EXPERIMENTS AND RESULTS

3.1 Q-Learning Results

Interpreting the results of the alpha variations of Q-Learning algorithms there is a very clear pattern:

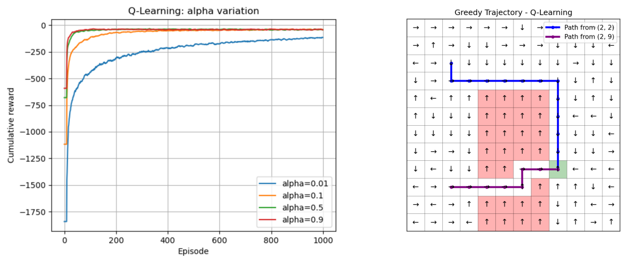

Figure 1: Q-Learning performance under different learning rates α in the ShortCut environment and the resulting greedy trajectory. Lower α values lead to slower learning, while higher values enable quicker convergence. The agent ultimately learns an aggressive trajectory that maximizes long-term rewards.

According to Figure 1, it is clear that a larger alpha results in a greater learning curve and higher cumulative rewards in earlier episodes. This is because α is the learning rate, and therefore a higher learning rate means that when a state-action value is updated more of the previous state-action value is overwritten. This results in a faster learning curve and a higher cumulative reward earlier in the training. On the other hand, the algorithm with a low alpha: $\alpha = 0.01$ had a very slow learning curve and yielded a lower cumulative reward after training.

Investigating the path generated by the Q-Learning algorithm in Figure 1 it is clear that it found the optimal path from both starting locations.

We can see that both the blue and purple lines end up at the end goal in a non-stochastic environment and given the information that every steps yields a constant reward it is also apparent that these two paths are optimal.

3.2 SARSA Results

When interpreting the results of the alpha variations of the SARSA algorithm there is a also a clear pattern very similar to Q-Learning:

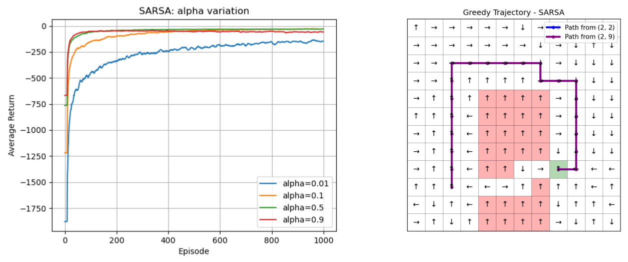

Figure 2: SARSA performance under different learning rates in the ShortCut environment (left) and the corresponding greedy trajectory after training (right). The learning curve shows that low α values result in slow convergence and reduced early performance. Compared to Q-learning, SARSA shows greater variance and slower learning due to its on-policy updates. The final greedy policy reflects a more conservative path, as the agent avoids risky areas for safer paths.

Although both Figure 1 and Figure 2 look very similar, there is a key difference, which is the end behavior of the alpha variations. Although the update equation of Q-Learning and SARSA is very similar, the key difference arises when selecting the next action. Q-Learning always calculates the updated state-action value using the maximum future rewards using $\max_{A'} Q(S', A')$ (off-policy), while in SARSA the next action A' is given by the epsilon-greedy policy (on-policy). This means that in the SARSA algorithm there is some randomness in updating the state-action value since the epsilon greedy policy has a chance of $\frac{\epsilon}{|A|-1}$ not selecting the best action. Therefore, while in Q-Learning the cumulative reward of each alpha variation approaches a certain value without variation after a number of episodes, in SARSA this is not the case. With a larger α, a greater portion of the state-action value is overwritten, and therefore in the $\alpha = 0.9$ model the approached value is lower since when a non-best next action is chosen, A', the future reward will be lower $Q(S', A')$, and therefore 90% of the original state-action value is overwritten to a lower value. On the other hand, with a smaller α a greater cumulative reward is approached since the action chosen by the policy has a smaller effect on the updated state-action pair.

Looking at the optimal trajectory given by the SARSA algorithm, it is clear that it has found a different path optimal compared to Q-Learning. The reason for this difference between Q-Learning and SARSA trajectories is the difference in the two algorithms. Q-Learning is off-policy, meaning that when updating the state-action values, it will always use the best action to calculate future rewards. On the other hand, as previously stated, SARSA is on-policy meaning that it will use the policy to calculate future rewards. This means that due to the randomness in the selection of the next action, it will prefer 'safer' routes where this randomness cannot set it back. This results in the algorithm preferring the long route because it ensures that it cannot 'fall down a cliff' due to the epsilon-greedy policy. Therefore, the main difference between SARSA and Q-Learning comes from the way of updating the state-action values; Q-Learning takes into account the best action, while SARSA uses the policy to generate an action.

3.3 Windy Environment

In addition to the normal ShortCut Environment, the performance of SARSA and Q-learning algorithms was compared on a Windy version of the grid. This environment adds stochasticity to the environment by adding downward wind 50% of the time after each action. Figure 3 shows the greedy trajectory produced by the Q-learning algorithm in the Windy environment. As it is an off-policy method, it is constantly updated with the highest action value, which leads to a more 'risky' path taken. When there is wind, this can result in high-risk and unstable trajectories.

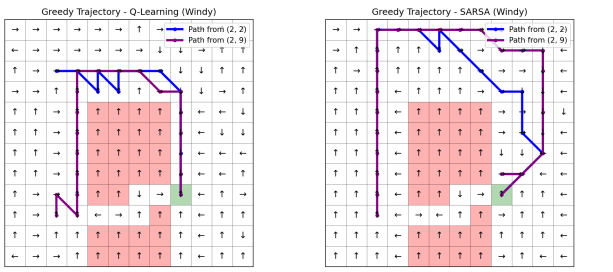
Greedy Trajectory - Q-Learning (Windy)
Greedy Trajectory - SARSA (Windy)
Path from (2, 2)
Path from (2, 9)

Figure 3: Greedy policies learned by Q-Learning and SARSA in the Windy ShortCut environment. Q-Learning favors high-reward but riskier paths, which become unstable under wind. SARSA, as an on-policy method, learns safer trajectories by incorporating actual actions during learning.

Figure 3 shows the policy learned by SARSA. Unlike Q-learning, SARSA takes into account the actual action taken by the ϵ-greedy policy, leading to a more careful path selection. The figure shows that SARSA is more suitable for environments with randomness, where stability is important. Q-Learning seems to try to maximize rewards at the cost of using the path near the cliff, high risk, whereas SARSA is more careful and takes fewer risks; however, it also obtains a path with a lower cumulative reward.

3.4 Expected SARSA

When analyzing the alpha variations of the expected SARSA algorithms in Figure 4, we can see a similar trend to the previous methods, where higher learning rates converge faster and in this case show a greater cumulative rewards within fewer episodes.

Like shown in Figure 4, the reason why $\alpha = 0.9$ offers the best performance is because, unlike SARSA, there is no randomness involved in updating the state-action values; therefore, a higher learning rate also offers high stability. This is due to the nature of the update equation previously discussed, which accounts for the weighted average future reward. This means that each possible next action is included, and therefore it is not necessary to choose a next action. However, in expected SARSA the approached cumulative reward is lower than Q-Learning because, like SARSA, it prefers the 'safer' path and accounts for the randomness in policy.

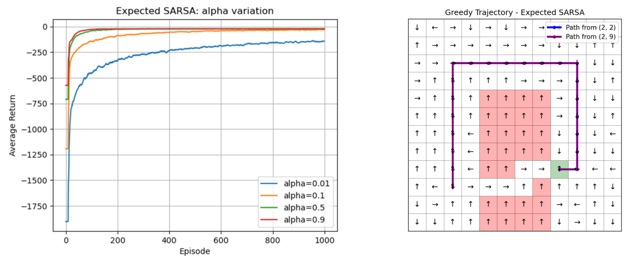

Figure 4: Expected SARSA performance across different learning rates (left) and the corresponding greedy trajectory after training (right). The learning curve shows that $\alpha = 0.9$ yields the best performance, with faster convergence and higher cumulative rewards. The resulting policy shows a balance between efficiency and safety, avoiding risky regions while maintaining relatively direct paths to the goal.

The greedy trajectory shown in Figure 4 shows that the agent is able to reach the goal state from both starting positions, in a similar manner to SARSA.

Compared to SARSA the main difference arises from the upper right corner which the two algorithms take differently but that is simply due to the randomness in SARSA due to the epsilon-greedy policy. Both the SARSA and expected SARSA paths yield the same cumulative reward, and they both prefer a path where no 'random error' can occur. However, compared to Q-Learning, the cumulative reward is less; however, since the algorithm is on-policy, it accounts for risk involving action selection, while Q-Learning prefers the risky but rewarding path through the cliffs.

3.5 n-step SARSA

Figure 5 shows us the impact of varying n on the agent's learning performance. For this plot, the value of α was fixed at 0.1, and different values of n are plotted.

In Figure 5 interesting pattern emerges: agents with multi-step updates ($n > 1$) result in higher cumulative reward earlier in the training. However, agents with a greater n also approach a smaller cumulative reward. This is due to the way the update function works in n-step SARSA. At multi-step updates the function collects the reward for n steps and updates the state-action value. The update equation includes information on future rewards. With a larger n the update function is more dependent on the collected reward after n steps to update the state-action values because the collected reward will be greater than with a smaller n. This results in more accurate results at the start and better short-term performance. However, a smaller n means that a more accurate future reward is needed since the reward after n steps is less. This results in algorithms with smaller n relying on more accurate Q-values. Thus, earlier in training, agents with a larger n preform better, but also converge to a lower cumulative reward.

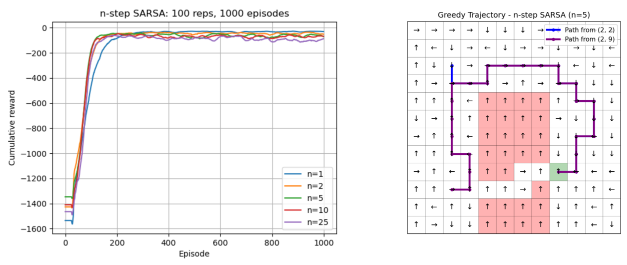

Figure 5: n-step SARSA performance for varying step sizes n (left) and the resulting greedy trajectory for $n = 5$ (right), using a fixed learning rate of $\alpha = 0.1$ Intermediate values such as $n = 2$ and 5 show the best balance between learning speed and stability. Larger n values lead to slower convergence. The policy for $n = 5$ trades between exploration and reward maximization, resulting in a relatively efficient and robust path to the goal.

When n is set to ∞ the algorithm will only use only the actual observed reward and will not account for any predicted reward. This means that when updating the state-action values it will simply use the observed discounted collective reward to update states-action values.

Looking at the path taken by the n-step algorithm, it is clear that it does not find the safest or most rewarding path. This is because of the real vs. predicted reward trade-off. The greater the n the greater the importance of real rewards and the smaller the importance of predicted rewards. Although with $n = 5$ the agent relies on future reward, since the agent must take 5 steps before updating, there might be some noise in the state-action values resulting in some approximation error and error in selecting the path. It is clear that the algorithm finds the goal state but it does not do in in the most rewarding or safest way.

4 DISCUSSION

4.1 Algorithm and Path Comparison

Across the different experiments, we have observed different learning behaviors and trajectories among the different algorithms.

The Q-Learning algorithm proved to consistently converge quickly, and found the most optimal paths in terms of rewards. This was also true in the Windy environment, where the Q-Learning algorithms showed the most direct route. However, this comes with the weakness that it may not choose the most robust path as it showed to perform riskier moves because it is off-policy.

The SARSA algorithm was slightly slower to converge when compared to the Q-learning algorithm. SARSA also showed a more conservative path selection. As it is an on-policy method, the algorithm learned to stay away from cliffs or windy zones.

Expected SARSA finds a balance between fast learning and stable behavior. The learning curves of expected SARSA were generally smooth and consistent.

Finally, n-step SARSA, showed a range of learning performance based on the value of n. The lower values of n showed that the algorithm was cautious in learning but found the path with the greatest cumulative reward. Intermediate values such as $n = 5$ achieved stable learning and efficient paths. However, a too large value of n proved to be inefficient.

4.2 Reflection

Through the implementation of four different model-free RL algorithms, we have gained insight into the difference in learning patterns for each algorithm and observed the different trajectories. We have observed that Q-Learning was the most aggressive learning strategy, where it systematically finds the shortest and most direct route, taking more risks in stochastic environments such as the Windy ShortCut Environment. SARSA, on the other hand, consistently showed more conservative behavior and illustrated how on-policy learning accounts for stochastic environments. Overall, through this assignment, we were able to deepen our understanding of the trade-offs between on-policy and off-policy learning algorithms. These insights are relevant in RL in general, as in real-world problems, agents must balance uncertainty and risk in stochastic environments.

5 CONCLUSION

In this assignment, we implemented four different model-free RL algorithms: Q-Learning, SARSA, Expected SARSA, and n-step SARSA. Each algorithm showed specific characteristics, with different strengths and weaknesses. Q-Learning showed efficiency and optimality on the path chosen, at the cost of taking risks in stochastic environments due to its off-policy nature. SARSA on the other hand showed a conservative approach due to its on-policy learning, which led to longer but safer trajectories. Expected SARSA extended SARSA by removing the randomness from the next action selection due to the epsilon-greedy policy and also preferred the longer and safer path. Lastly n-step SARSA incorporated a balance between rewards observed during n steps and also future rewards. This showed good results when n is adjusted to an appropriate value. By taking into account a sequence of n rewards, n-step SARSA reduces some of the bias present in the default SARSA, which heavily relies on immediate rewards and bootstrapped estimates of future values.

We can conclude that each algorithm has strengths in different areas, and the choice of a specific algorithm depends on the nature of the environment, and desired performance. While off-policy agents find the path with the greatest cumulative reward they might not be suitable for policies with large exploration-exploitation trade-offs.For future research, it would be interesting to tests these algorithms in various partially observable environments, or to analyze more in depth the effect of n values in n-step SARSA algorithms.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.