Assignment 2: Model-free reinforcement learning

Simone de Vos Burchart (s1746995) & Hao Chen (s3990788)
Group Number: 80

1 Introduction

In reinforcement learning (RL), the goal is to train an agent that can interact with the environment and maximize its cumulative reward. The agent makes decisions by selecting actions based on the current state and learning over time which actions lead to the most favorable outcomes. And one of the central challenges in RL is how to find a balance between exploration (trying different actions to discover better rewards) and exploitation (selecting actions that have already led to high rewards).

In this paper, we focus on two model-free RL algorithms: **Q-Learning** and **SARSA** (State-Action-Reward-State-Action). Both algorithms aim to estimate the value of state-action pairs, but they differ in how they update their estimates and explore vs. exploit.

We conduct experiments in a 12x12 grid, where the agent must navigate toward the goal while avoiding cliffs. In this environment, each step the agent takes gives a reward of -1. But if the agent walks off a cliff, it receives a reward of -100 and returns to the starting point. Reaching the goal does not provide any reward, but it ends the current episode. Thus, the agent must learn to reach the goal while maximizing the total cumulative reward.

By comparing the movement paths of different agents, we will evaluate their learning capabilities. We will also investigate how different parameters (such as learning rate and exploration rate) affect the performance of each algorithm. Throughout this paper we will use the values ϵ=0.1 and γ=1, and α=0.1 unless stated otherwise.

2 Q-Learning

The first model-free RL algorithm we tried was Q-Learning, an off-policy Temporal-Difference control algorithm.

2.1 Methodology

The Q-learning algorithm is defined by the update function:

$$Q(S_t, A_t) \mathrel{+}= \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1,a}) - Q(S_t, A_t)\right] \tag{1}$$

where $Q(S_t, A_t)$ is the learned action-value function applied to state S_t and action A_t, R_{t+1} is the observed reward for taking action A_t in state S_t, α is the learning rate, γ is the discount parameter, and $Q(S_{t+1}, a)$ is the Q-value for taking action a in the next state S_{t+1} [1]. With this algorithm, Q will converge to the optimal action-value function q_*.

It selects actions with an ϵ-greedy policy, defined as

$$\pi_\epsilon(a) = \begin{cases} 1-\epsilon, & \text{if } a = \text{argmax}_{b\in\mathcal{A}} Q(b) \\ \frac{\epsilon}{|\mathcal{A}|-1}, & \text{otherwise} \end{cases} \tag{2}$$

where $\mathcal{A}$ is the set of actions. It is trained over a number of episodes by trying actions until it reaches the goal, while updating Q.

2.2 Result

We first ran a single experiment with 10000 episodes. The best actions as learned by the agent, and the greedily chosen path, are shown in Figure 4a. Figure 4a shows that both paths successfully reach the goal with the fewest possible steps and without falling off a cliff, leading to the highest possible reward.

Next we ran experiments with different values of α from the set $\{0.01, 0.1, 0.5, 0.9\}$. For each α we ran 100 repetitions of the same experiment, with 1000 episodes per repetition. We computed the average over the repetitions and applied smoothing. The resulting reward curves are shown in Figure 1. The graph shows that in the earliest episodes, the reward is very negative, meaning a lot of steps or falling off cliffs. As the number of episodes increased, the average reward increased, meaning the agent had learned to avoid cliffs and to seek shorter paths. The agents have different results depending on the learning rate α. For the lowest value, α=0.01, the average reward started much lower in the first episodes, and as α increased, the rewards in the first episodes were higher. They all showed a very steep learning curve early on, with α=0.5 and α=0.9 behaving very similarly and leveling at about -50 after less than 50 episodes, and for α=0.1 after about 400 episodes. α=0.01 on the other hand, was still increasing its average reward at 1000 episodes. These results show that the higher the learning rate, the faster the agent was able to learn the values of the different actions and maximize its reward.

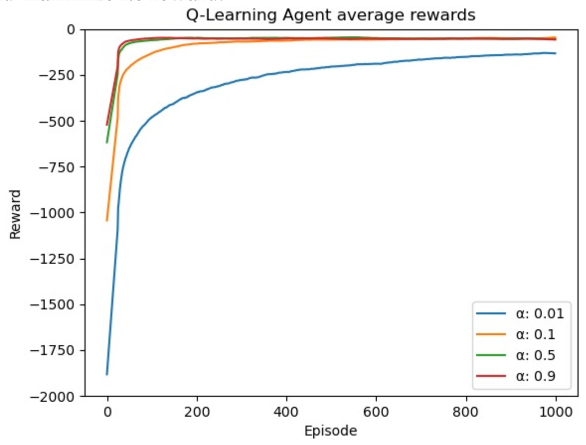


Figure 1: Reward curves over 1000 episodes averaged over 100 experiments and smoothed, for Q-Learning agents with various learning rates α, and fixed parameters ϵ=0.1 and γ=1.

3 SARSA

The second model-free RL algorithm is SARSA. The main difference between SARSA and Q-learning is that SARSA is an on-policy algorithm, which updates the Q-value of the corresponding state-action pairs based on the currently selected action, whereas Q-learning is an off-policy algorithm whose Q update depends on the maximum Q value chosen by the greedy policy, regardless of the current action.

3.1 Methodology

The SARSA algorithm is defined by the update function:

$$Q(S_t, A_t) \mathrel{+}= \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{3}$$

where $Q(S_{t+1}, A_{t+1})$ is the Q-value for taking action A_{t+1} in the next state S_{t+1} and the rest is defined as in Section 2.1 [1]. Since SARSA is an on-policy algorithm, the update formula uses the ϵ-greedy policy as defined in equation (2) to select the next action, and does not use the maximum Q-value of all possible actions (like Q-learning).

3.2 Result

Again, we ran a single experiment with 10000 episodes. As before, the actions with the maximum Q values for each state and the greedily chosen path are shown in Figure 4b.

By comparing Figures 4a and 4b, we can clearly see that the SARSA agent makes different choices when avoiding the cliff compared to the Q-learning agent. When the SARSA agent's starting position is at the bottom, it does not choose the shortest path (which passes through the cliffs), but instead selects a longer route that avoids the cliff area in the middle. In contrast, the Q-learning agent chooses the path that goes through the middle of the cliff, which, although shorter, might lead to the agent falling off the cliff. This difference arises because Q-learning updates Q-values based on the highest action-value for the current state (it would not consider that the next action might lead to cliffs), while SARSA considers both the current and next actions, making it more cautious and inclined to choose safer paths.

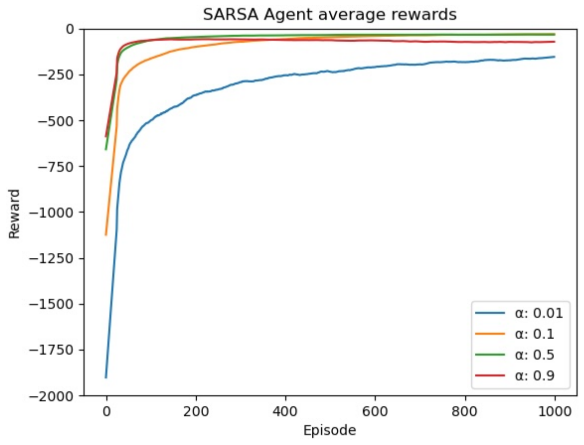
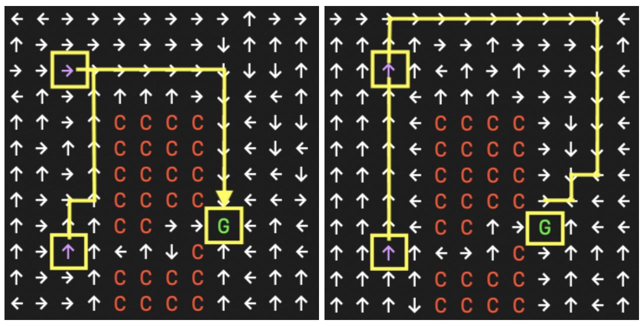


Figure 2: Reward curves over 1000 episodes averaged over 100 experiments and smoothed, for SARSA agents with various learning rates α, and fixed parameters ϵ=0.1 and γ=1.

In addition to the path choices observed, the learning rate α also plays a critical role in the agent's learning behavior. Figure 2 illustrates the performance of the SARSA agent with varying α values, showing the cumulative reward over episodes for different learning rates. Based on Figure 2, as α increases, the agent's learning speed accelerates. Before episode 100, a larger learning rate allows the agent to learn faster and achieve better rewards. However, when the learning rate becomes too large, the reward curve exhibits more fluctuations. For example, for α=0.9, the SARSA agent's rewards start to decrease as the number of episodes increases. On the other hand, a very low learning rate, such as α=0.01, results in a slower learning process but with less reward fluctuation. From the graph, we can observe that when α=0.5, the SARSA agent performs the best overall. It reaches an average reward of -75 before episode 100, and the reward curve continues to rise gradually as the episodes progress.

4 Stormy Weather

We also ran two experiments with the previous two agents in a slightly different environment, where after every step, there is a 50% chance that there is a wind that pushes the agent down one square.



(a) Q-learning (b) SARSA

Figure 3: The actions with the maximum Q values for each state represented by the direction of the arrow, for the Q-learning agent (a) and the SARSA agent (b) both trained for 10000 episodes with parameters α=ϵ=0.1 and γ=1. The yellow lines mark the actions that a greedy policy would select to go from the starting points (in purple) to the goal (the green G), if there was no wind. The red Cs represent the cliffs.

For both the Q-learning agent and the SARSA agent, we ran an experiment with 10000 episodes. The actions with the maximum Q values for each state are shown in Figure 3a for the Q-learning agent and in Figure 3b for the SARSA agent. The greedily chosen paths are marked in yellow. However, this assumes that there is never any wind. In reality, the path of the agent depends on the stochastic nature of the wind. This can be seen when comparing Figures 3a and 3b about the windy grid with Figures 4a and 4b about the non-windy grid. Both the Q-learning agent and the SARSA agent learned to take a path that goes further around the cliffs in the windy environment, to avoid the risk and the potential cost of being blown off a cliff. Like in the non-windy environment, the SARSA agent learns a path that is longer but safer than the path of the Q-learning agent. However, safety is not the only way the wind impacts the agents. As can be seen in 3b, the SARSA agent actually depends on the wind to reach the goal.

If the wind were to change direction, the agents would learn the path that maximizes the cumulative rewards given that wind. The path through the middle of the cliffs would likely not be chosen, because for every wind direction there is at least one state where there is a 50% chance of being blown off a cliff. If the wind came from the bottom, the paths for both agents might be similar to the non-windy environment as shown in Figures Figures 4a and 4b, except for the Q-learning agent's path through the middle of the cliffs. This is because apart from the path through the middle, wind from the bottom can never result in falling of a cliff. If the wind came from the left, the agents probably would start out more towards the left, and if the wind came from the right, on the side of the goal they would probably learn a path that stays more on the right.

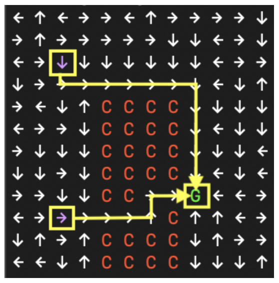
5 Expected-SARSA

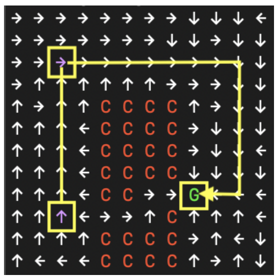
5.1 Methodology

The expected-SARSA is defined by the following update function:

$$Q(S_t, A_t) \mathrel{+}= \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \quad (4)$$

with the variable defined as before and where the term $\sum_a \pi(a|S_{t+1})Q(S_{t+1}, a)$ is the expected Q-value for the next state


(a) Q-learning


(b) SARSA

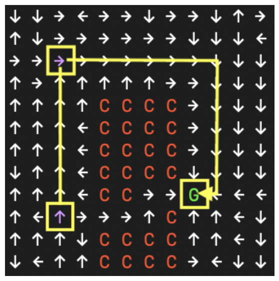

(c) Expected SARSA

Figure 4: Actions with the maximum value for each state according to the various agents, trained with 10000 episodes, with fixed values α=ϵ=0.1 and γ=1. The path to the goal from each starting point that a greedy policy would take is marked in yellow.

S_{t+1} [1]. It calculates the expected value with the probability of actions based on the policy. Since it used an ϵ-greedy policy to choose actions, the probability of the action with the maximal Q-value is 1-ϵ, and the probabilities of other three actions are $\epsilon/3$.

The expected SARSA update adjusts the Q-value based on the observed reward and the expected future Q-values for all possible actions in the next state. This is unlike SARSA, where the next action's Q-value is updated based on the action actually taken. Instead, expected SARSA uses the expected value of all actions in the next state, weighted by the policy probabilities.

5.2 Result

In the experiment, we ran a single trial in the ShortCutEnvironment with 10000 episodes and set $\alpha = \epsilon = 0.1$. The actions with the highest Q-values for each state are shown in Figure 4c. The yellow lines mark the actions that a greedy policy would select to move from the starting positions (indicated in purple) to the goal (marked as the green G).

By comparing Figures 4a, 4b, and 4c, we observe that the Expected-SARSA agent makes different choices compared to both Q-learning and SARSA agents when avoiding the cliff. Specifically, the Expected-SARSA agent also avoids cliffs, but doesn't choose to take the long way around like SARSA, which is relatively more optimized for path selection. This is because in Expected SARSA, when updating the Q-value, the weighted average Q of all possible actions in the future state is computed, whereas SARSA updates the Q-value by considering only the current state and the immediate rewards of the selected actions, which means that SARSA's learning is more dependent on the paths actually experienced. Since SARSA only updates its Q-value based on the currently selected action, it may choose a path because the reward of a certain step is particularly high, which may cause it to fall into a local optimum early and stop exploring better paths. In contrast, Expected SARSA considers the expected Q-values of all possible future actions at each update step, which makes it more likely to explore different paths and ultimately choose those that are less risky and more rewarding.

From Figure 5, we can observe that with learning rate α=0.01, the Expected SARSA agent's average reward starts at around -1800, which is the lowest average reward in the graph. We can see that when the learning rate is very low, the agent's learning speed is quite slow, and after 100 repetitions of 1000 episodes, the average reward reaches around -200. There are two notable lines in the graph corresponding to α=0.5 and α=0.9. These lines almost overlap, with the α=0.9 line showing faster learning and higher rewards until episode 200. However, after episode 200, the rewards of the two lines converge. Therefore, we can conclude that when α is greater than 0.5, a higher learning rate does not lead to better rewards for the agent in the end.

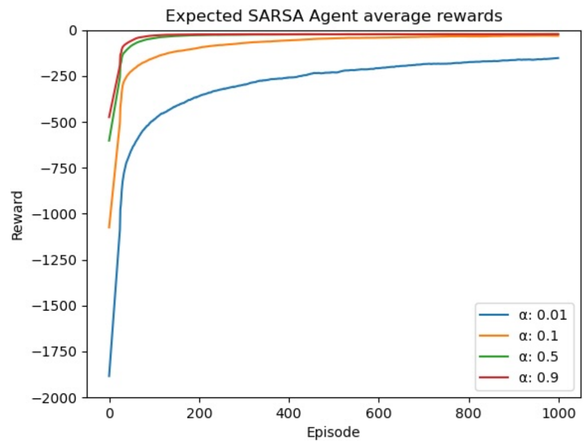


Figure 5: Reward curves over 1000 episodes averaged over 100 experiments and smoothed, for Expected SARSA agents with various learning rates α, and fixed values ϵ=0.1 and γ=1.

6 n-step SARSA

6.1 Methodology

The n-step SARSA is defined by the following update function:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} \tag{5}$$

$$Q(S_t, A_t) \mathrel{+}= \alpha \left[G_{t:t+n} + \gamma^{n-1} \cdot Q(S_{t+n}, A_{t+n}) - Q(S_t, A_t) \right] \tag{6}$$

where $Q(S_t, A_t)$ is the learned action-value function applied to state S_t and action A_t, α is the learning rate, R_{t+1} is the observed reward for taking action A_t in state S_t, γ is the discount factor, $G_{t:t+n}$is the cumulative payoff from time step t to t+n, which represents the total payoff computed from the current moment through the Q-values of future rewards and state-action pairs, and $\gamma^{n-1} \cdot Q(S_{t+n}, A_{t+n})$ denotes the value of Q from the t+n step, representing the value of the future state-action pair [1].

In the n-step agent, the current state and the subsequent n steps' states, actions, and rewards are stored together for future updates. The update() function is used to update the Q-values, where all actions are selected based on the ϵ-greedy policy. After each update, only the Q-value corresponding to the current state-action pair is updated. Once the Q-value is updated, the most recent state, action, and reward are popped out of the list, and the agent continues with the next state and action for further updates.

6.2 Result

From Figure 6, it is clear that the learning process of the n-step SARSA agent shows different characteristics depending on the value of n. In the experiment, we set different values for n: 1, 2, 5, 10, and 25, and ran 1000 episodes, doing 100 repetitions. The figure shows the average reward curves for each value of n.

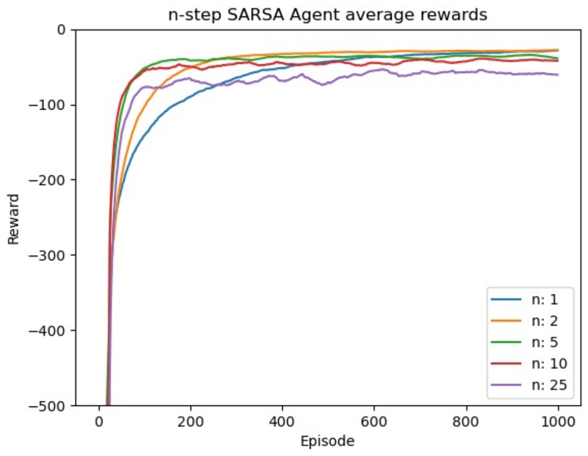


Figure 6: Reward curves over 1000 episodes averaged over 100 experiments and smoothed, for n-step SARSA agents with various values of *n* and α=0.1,and fixed parameters ϵ=0.1 and γ=1. It is zoomed in for more detail.

Figure 6 does not show it, but for smaller values of n (n=1, n=2, n=5), the starting point of the average reward is highest, around -900. As n increases, the starting point of the average reward decreases, with n=25 reaching approximately -1850. All curves for different values of n show rapid learning progress in the initial episodes, indicating that during this phase, the agent quickly adjusts its strategy to avoid falling off a cliff and moves toward the goal. The curves for different values of n display a similar upward trend during this phase.

If we look at Figure 6, we can observe the agent's learning results over the long term. We can see around episode 100, that the line for n=5 reaches an average reward around -50 before the others. Below that is n=10, which also reaches -50 quickly, but with significant fluctuations in subsequent learning. n=25 shows similar behavior, rapidly reaching a higher average reward (around -90), but with large fluctuations and no stable increase. The n=2 line, while not reaching a good average reward quickly, shows a steady increase and surpasses the n=5 line at around 300 episodes. The n=1 line increases slowly, but as the number of episodes increases, its average reward surpasses that of n=10 and n=25. This could be because n=1 focuses on immediate rewards and learns more conservatively, allowing it to avoid risky or less optimal paths early on. Although it starts slowly, over time, its steady learning process allows it to catch up with, and eventually surpass, the more volatile learning behavior of n=10 and n=25, which take longer to stabilize and are more affected by long-term returns. If n increases, the computational time increases, as the agent needs to store and process more steps before updating. If n -> ∞, the algorithm essentially becomes a Monte Carlo method [1], where the agent updates the Q-values based on the total return for the entire episode. This can result in delayed learning and infrequent updates.

7 Comparison and conclusions

We calculated the average cumulative reward of each single-step agent with the different values of α for the last 100 episodes, to eliminate the effect of the early learning curve. The results can be seen in Table 1.

α	Q-learn.	SARSA	Exp. SARSA	n	n-st. SARSA
0.01	-136.7471	-159.4436	-158.4668	1	-30.1570
0.1	-52.2423	-31.4573	-31.4462	2	-28.1248
0.5	-52.9119	-33.1088	-23.1724	5	-35.1292
0.9	-52.6274	-76.2872	-22.7701	10	-43.7906
				25	-51.6580

Table 1: Comparison of the average cumulative reward of 100 repetitions of various agents with different parameters, over the last 100 of 1000 episodes, and fixed parameters ϵ=0.1 and γ=1.

Table 1 shows that the Expected SARSA agent outperforms the SARSA agent at every tested learning rate. That is because the expected SARSA agent is using the weighted average of all possible future actions to update the Q-value, and the SARSA agent updates the Q-value with the action by the current state, which means the SARSA agent is more likely to prematurely choose an action with a high Q-value, potentially leading it into a local optimum. It also shows that the best SARSA agent outperforms the best Q-learning agent, which can be explained by the safer path the SARSA agent learns compared to the Q-learning agent. The Expected SARSA agent with α=0.9 also has the advantage of learning much faster than the best of the other two single-step agents, as seen in Figures 1, 2, 5.

Comparing the single-step agents with the n-step SARSA, we see that at a low value of *n*, the n-step SARSA agent outperforms the SARSA agents, and it outperforms the Q-learning at any tested value of *n*. The 1 and 2-step SARSA agents outperform the Expected SARSA agent when they both use learning rate α=0.1. That is because Expected SARSA needs a higher learning rate to update the Q-values.

It is clear that the Expected SARSA agent is better than the other single step agents. It shows the best results at a high learning rate. This leads us to the question: would an n-step SARSA agent also perform better at a higher learning rate, and would it be better than the Expected SARSA agent at the same learning rate? Furthermore, future experiments might look into combining the 2-step SARSA agent with the Expected SARSA agent, to see if this performs better than the single step Expected SARSA agent.

8 Conclusion

Through the experiments described above, we found that Expected-SARSA outperformed the others because the future Q-values of the current state are calculated by considering the weighted average of all possible actions in the next state, based on the action selection probabilities. Thus it will choose safer and more optimized paths. N-step SARSA performed better than regular SARSA because it incorporates multiple steps in the return calculation, helping to balance immediate and future rewards, making the learning process faster and more stable. SARSA, relying on immediate rewards, sometimes leads to suboptimal decisions, while Q-learning, due to its off-policy nature, is slower to respond to environmental changes and does not avoid paths that might lead to the cliff, as it only updates Q-values based on the highest-value action.

9 Appendix

In Section 4 - Stormy Weather, we added wind occurrences (WindyShortcutEnvironment) to explore whether this would affect the agent's movement path. The wind in the environment pushes the agent down by one square with a 50% probability, introducing uncertainty in the agent's navigation.

In this section, the main question we aim to explore is how changes in the probability of wind occurrences affect the agent's learning behavior, particularly its movement path. Considering that Q-learning is an off-policy algorithm, its decision-making process primarily focuses on maximizing the action-value for each state rather than considering state transitions. Theoretically, the probability of wind occurrences should not significantly affect the behavior of the Q-learning agent. We have also tested this, and the paths did not change significantly; the paths kept the same as with a 50% probability wind(3a). Therefore, we will focus on testing the impact of wind probability on the movement path of the SARSA agent.

9.1 Hypothesis

SARSA is an on-policy algorithm, meaning it learns and updates Q-values based on the actions actually taken and the resulting state transitions. Therefore, we hypothesize that SARSA will be more sensitive to changes in wind probability. When the wind probability is low, the agent may adjust its strategy to avoid disturbances caused by the wind, choosing safer paths. However, in environments with higher wind probabilities, the agent needs to adapt its strategy and learn how to incorporate the wind's effect into its path selection, potentially choosing longer, safer routes to avoid being blown off the cliffs.

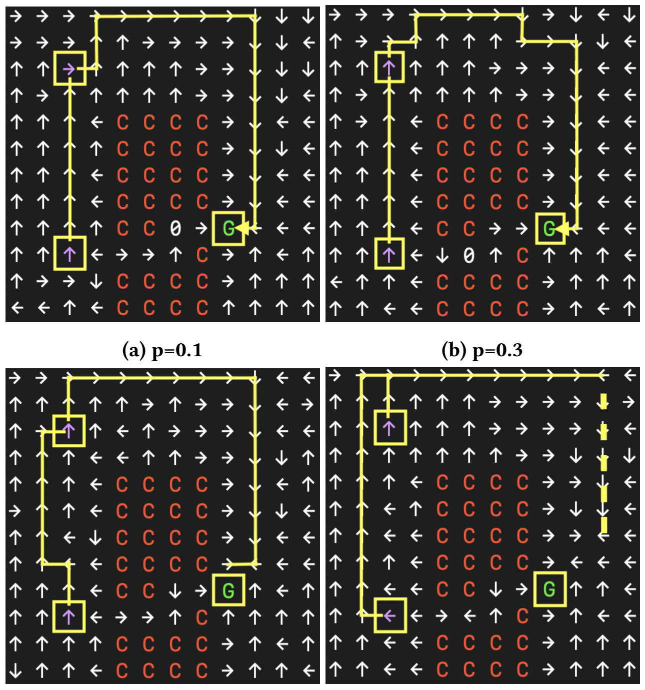
9.2 Experiment

We created a new class called differenceWindyShortcutEnvironment, which inherits from WindyShortcutEnvironment. In the new class, we modified the step() function to introduce windProbs, which determines when the wind occurs. Specifically, the wind will push the agent down one square with a certain probability, introducing uncertainty into the agent's movement.

For each wind probability p, we run a single experiment with p and track the agent's performance over 10000 episodes. After the 10000 episodes, we use the render_greedy() method of the environment to print the agent's path. We then observe all the paths to determine if the agent adjusts its strategy to account for wind disturbances.

9.3 Result

Figures 7 (a to d) illustrate the agent's paths for various wind probabilities (p=0.1, 0.3, 0.5, 0.7). Figure 7a shows that when the wind probability was low (p=0.1)SARSA agents mainly followed the optimal path, avoided cliff areas, and chose relatively short paths. When wind probability increased (figure 7 b-d) the agent adapted by choosing longer paths to avoid the potential disruption caused by the wind. In the cases where the wind probability is p=0.5 and p=0.7, we can observe that the agent starts utilizing the wind to help reach the goal. This is significant in Figure 7d, where the agent consistently moves left and right at the top. This behavior shows that the agent is waiting for the wind to blow it downward. It avoids the agent moving downward by its own and potentially getting closer to the cliff. As mentioned in section 4, when the wind probability is 50%, the SARSA agent also uses the wind to reach the goal while avoiding the cliff.


(a) p=0.1 (b) p=0.3

(c) p=0.5 (d) p=0.7

Figure 7: The actions with the maximum Q values for each state represented by the direction of the arrow, for the SARSA agent at different wind probabilities p, trained for 10000 episodes with parameters α=ϵ=0.1 and γ=1. The yellow lines mark the actions that a greedy policy would select to go from the starting points (in purple) to the goal (the green G), if there was no wind. The red Cs represent the cliffs.

9.4 Conclusion

This simple experiment shows the capability of the SARSA agent to adapt to its environment and find an optimal policy, not only despite the sometimes difficult conditions that the environment presents, but at times precisely thanks to those conditions. This makes SARSA a good candidate for learning in more complex environments, even those approaching real-world complexities, like the humanoid running experiment we were shown in the first lecture of RL. However, we have seen that the Expected SARSA agent outperformed the SARSA agent in the experiments without wind. Future research should look into the performance of the Expected SARSA agent in various wind conditions and compare it to SARSA. It would also be good to add more complexities to the environment to see how well the agents are able to adapt to those.

References

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