Assignment 2: Model-free RL

Group Kacper Nizielski (s4068858) & Emmanouil Zagoritis (s4076893)
Group Number: 60

INTRODUCTION

A Model-free reinforcement learning algorithm finds the best way to move without a model of the environment and get the best reward possible. It learns the effect of its actions through experience, using trial and error along with various formulas. The algorithm regulates its policy for optimal reward, by trying an action multiple times and checking from results if it leads to a better reward. In this assignment, we implement and compare four different model-free RL methods: Q-learning, SARSA (State–Action–Reward–State–Action), Expected SARSA, and n-step SARSA, as described in Sutton and Barto's book [1].

Agents move in a given custom environment ShortCut. The ShortCut environment is a 12x12 grid where an agent can move to squares next to the current one. The world has a terminal which is a goal square. The agent starts in one of two squares, each has an equal probability. The grid also contains cliff squares with a reward of -100 points and returns to the starting point if the agent lands on this square. All the other squares have a reward of -1. The goal of each agent is to maximize the cumulative reward of each episode.

Our assignment aims to check each algorithm's behavior in the ShortCut environment. We intend to evaluate the performance of every algorithm, especially their learning speed and how well the policies work / the effectiveness of learned policies. Particularly, we are interested in the time and reliability of convergence of each method to almost optimal policy.

Firstly, we will introduce and describe the methodology for every agent, including the equations. After that, we show the results of different experiments for various repetitions and number of episodes. Each experiment will be provided with a graph visualization. The interpretation for each experiment will also be provided. We will also experiment on a stochastic environment, such as by adding wind to the environment described, and present the results of that change. In the end, we will compare the best version of every algorithm, by discussing their strengths and weaknesses.

Q-LEARNING ALGORITHM

It is an off-policy temporal difference reinforcement learning algorithm, that estimates the value of taking a certain action in a given state, and updates this value, using the greedy action. The Q-learning algorithm uses the maximum estimated Q-value of the next state $\max_a Q(S_{t+1}, A)$, which means that it always allows the agent to act optimally from the next step. In each step, the Q-value is updated by using this formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, A) - Q(S_t, A_t)]$$

where Q is the Q-value, S is a state at a timestep t, A is an action at a timestep t, α is the learning rate, γ is the discount factor, R_{t+1} is the reward received after taking action A_t in state S_t and $\max_a Q(S_{t+1}, a)$ is the estimated best future value. In the update function, the Q-value is updated by adding the difference between the expected future reward and the current Q-value and adding the observed reward to the Q-value for the current action in the current state.

To test the algorithm we ran an experiment where we checked the effect on the performance of the Q-learning agent, by changing its learning rate (α) to different values in 1000 episodes, averaged by 100 repetitions. As Figure 1 shows the used values are 0.01, 0.1, 0.5, and 0.9.

At first, the average rewards for agents were low because of exploration. From the image, it can be seen that agents with higher learning rates improved faster than those with smaller α. The agents with $\alpha = 0.5$ and $\alpha = 0.9$ show approximately the same performance. Although the agent with $\alpha = 0.1$ falls slightly behind in the early episodes compared to the two highest α values, it reaches similar Q-values in the final episodes. The agents with $\alpha = 0.01$ stand out from other agents, as it learns much slower, and average rewards remain lower until the last episode. Overall, Figure 1 shows that agents with higher learning rates learn much faster than those with low learning rates.

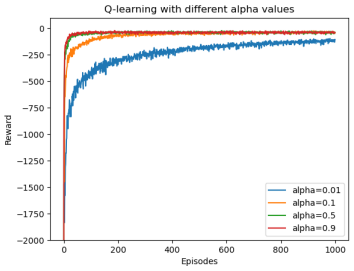

Figure 1: Q-learning performance over 100 repetitions of 1000 episodes for different learning rates (α), using $\epsilon = 0.1$, and $\gamma = 1.0$. Higher α values lead to faster convergence and better average rewards.

Figure 2 illustrates the greedy paths of the Q-learning agent. It can be seen that Q-learning takes hazardous paths, one near the cliffs and the other through the passage in the cliffs. Although it is risky to choose these paths, as the agents can step on the cliff position and get a negative reward, they are the most optimal ones to reach the goal.

SARSA ALGORITHM

SARSA is an on-policy temporal difference reinforcement learning algorithm that, similarly to Q-learning, on a specific given state based on the action taken updates the Q-values to maximize future rewards.

This algorithm is on-policy, meaning that its Q-values are updated by considering the next action A_{t+1}, which is taken at the next state S_{t+1}. Specifically, the Q-value is updated with the following equation:

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

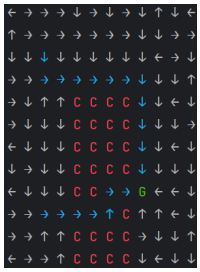

Figure 2: Greedy paths visualization for the Q-learning agent. Arrows indicate the optimal action at each state, with blue arrows showing the greedy path from both starting positions, avoiding cliffs and demonstrating good final policy performance.

where $Q(S_{t+1}, A_{t+1})$ is the Q-value of the next state-action pair, selected with ϵ-greedy policy, and in case there is not next state it is 0. In the update function, the Q-value is updated by adding the difference between the observed reward plus the discounted Q-value of the next-action pair and the current Q-value to the Q-value for the current action in the current state.

Same as with Q-learning, we tested the algorithm by running an experiment using different alpha values. The tested α values were again 0.01, 0.1, 0.5, and 0.09. As shown in Figure 3, agents with learning rates $\alpha = 0.5$ and $\alpha = 0.9$ improve more quickly on early episodes than the other agents. At about the end of the 1000 episodes of the training, agents with learning rates $\alpha = 0.1$ and $\alpha = 0.5$ slightly surpass the agent with learning rate $\alpha = 0.9$. Meanwhile, the agent with the very low learning rate of $\alpha = 0.01$ constantly performs worse than the rest. These observations indicate that medium and high learning rates result in better average rewards in early episodes and after a while, the smaller and medium learning rates perform better than the high learning rates by achieving slightly higher average rewards at the end of the episodes.

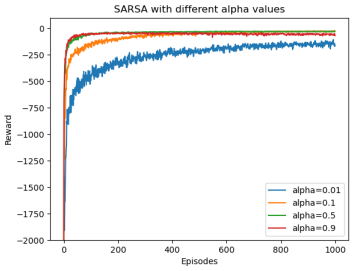

Figure 3: SARSA performance over 100 repetitions of 1000 episodes for different learning rates (α), using $\epsilon = 0.1$, and $\gamma = 1.0$. Initially, medium and high α values achieve better average rewards, while at the end, small and medium α values slightly outperform the high α value on average rewards.

We visualize the behavior of the SARSA agent in the ShortCut environment. Figure 4 illustrates the greedy paths of the SARSA agent. We observe that the agent chooses a safer path avoiding getting close to the cliffs. This behavior is explained by the on-policy characteristic of the algorithm. Specifically, as mentioned before the Q-values are updated based on the actual actions taken, which results in avoiding the negative rewards from the cliffs.

Figure 4: Greedy paths visualization for the SARSA agent. Arrows indicate the optimal action at each state, with blue arrows showing the greedy path from both starting positions. It prioritizes safety by taking safe and long routes, successfully avoiding the cliffs.

The difference between SARSA and Q-learning algorithms is their policy approach, as SARSA is on-policy while Q-learning is off-policy. We notice that because of this difference, SARSA has a more stable but slower convergence, while Q-learning converges faster but with more variability. Additionally, the policy difference can be observed from the paths, with SARSA choosing more stable and safe routes over risky shortcuts near the cliffs that Q-learning chooses.

STOCHASTIC ENVIRONMENT

Afterwards, the new Windy ShortCut Environment was introduced to conduct another experiment. The environment is very similar to the ShortCut environment but the new feature was added: stochastic wind. It is about the agent having 50% chance of moving one square below, after every action.

(a) Greedy path of Q-learning algorithm, using $\alpha = 0.5$.

(b) Greedy path of SARSA algorithm, using $\alpha = 0.5$.

Figure 5: Greedy paths visualization trained using Q-learning and SARSA in a windy windy environment. Q-learning chooses riskier but more optimal and better rewarded paths near cliffs, while SARSA tends to take a safer route to avoid the cliffs, showing its on-policy nature, leading to potential future punishments.

In this experiment, both Q-learning and SARSA algorithms were checked in the new environment. As illustrated in Figure 5 the paths are created as the algorithms make decisions using different methods.

While the Q-learning algorithm shown in Figure 5a learns paths reaching the goal, its routes tend to be riskier, going close to the cliff areas. It might be because of Q-learning's off-policy manner to always choose the best possible action. Furthermore, as it is more dangerous to choose these paths, it gives a higher cumulative reward.

On the other hand, the SARSA algorithm, shown in Figure 5b, selects the more cautious route, that omits the cliff areas. However, it results in not reaching the goal, due to SARSA's update method, which leads to a failure. Hence, although it chooses safer paths, it fails in its task.

Compared to the previous experiment where the ShorCut environment was used, the results changed in some way. The Q-learning algorithm in both experiments 2 and 5a finds a path, and it can be seen that these paths are optimal. The figures also illustrate, that the algorithm in the Windy ShortCut environment avoids choosing the path through the passage, as it used to in the Shortcut environment, as the wind would blow the agent on the cliffs, which would result in a worse final reward. However, the SARSA algorithm differs a lot in the Figure 4 from 5b. The agent in the ShortCut environment finds a cautious path that reaches the goal, whereas the agent in the new environment does not, showing that stochastic wind has a significant impact on SARSA's performance. In conclusion, Q-learning choosing a more risky but optimal path, finishes in goal, whereas SARSA, by choosing a safer route to not step on the cliff areas, is not even close to the goal.

In an event, where the wind would from a different direction, the agents would react differently depending on the wind's direction. Q-learning would try to find the new optimal paths based on maximum expected reward, by taking riskier routes. On the other hand, SARSA would adapt based on its current action, leading to more cautious and slower routes. Depending on the direction of the wind, there is also a possibility that the algorithm, especially SARSA, would struggle to find a way to reach the goal.

EXPECTED SARSA ALGORITHM

Expected SARSA algorithm that originates from Q-learning and SARSA. It is different from the other algorithms by updating its policy using the expected value of the next action. The update rule for this algorithm is described by:

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

where $\pi(a|S_{t+1})$ is the probability of taking action a in state S_{t+1} under the current policy.

Expected SARSA algorithm computes value of the next state by calculating an average over all possible actions. The algorithm updates its values deterministically, choosing the same direction as SARSA would choose in expectation. However, expected SARSA performs better than SARSA algorithm, because it has less variability, as it removes randomness from the next action.

As in the previously presented algorithms, we conducted a similar experiment with Expected SARSA over 1000 episodes, averaged across 100 repetitions, with different α values to find the method with the best final cumulative reward. We used the same four different values of α as before. The results are shown in Figure 6.

The plot indicates that an agent with $\alpha = 0.01$ shows the worst outcome as its final reward approximately converges to -250. In contrast, the other agents' performance was better, having the final reward around -70. The agent with the fastest learning and the best reward is with $\alpha = 0.9$, showing also not as much variation as the others. In addition, the agent with the second highest alpha value showed very similar performance to the first highest alpha with slower learning at the beginning. The agent with $\alpha = 0.1$ showed moderate learning speed. It is a little worse than 0.9 or 0.5 but is much better than 0.01, which varies a lot. In the end, it also converges to a final reward of around -70.

The plot shows a lot of similarity between the Expected SARSA algorithm and the Q-learning and SARSA algorithms. Higher learning rates result in better rewards and better learning performance. On the other hand, lower α values show the worst performance in every algorithm. They have the worst outcomes and vary a lot even in the last episodes. However, as in the experiment using the SARSA algorithm, the $\alpha = 0.5$ shows better performance in the last episodes, wherein the Figure 6 it can be seen that both learning rates are on the same level in the end.

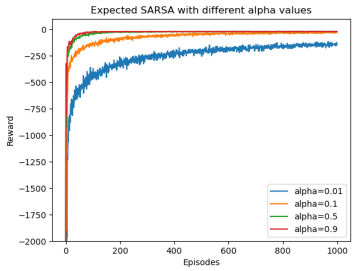

Figure 6: Expected Sarsa performance over 1000 episodes for different learning rates, using $\epsilon = 0.1$ and $\gamma = 1.0$. Higher α values result in faster learning and better average rewards, while lower values lead to worse performance.

As described earlier, Figure 7 illustrates greedy paths created by the Expected SARSA algorithm. The image indicates that the algorithm still chooses the safer path, avoiding the cliff areas, like SARSA. However, it takes a more optimal path, unlike SARSA. Instead of going near the border, which is a very cautious path, it only omits one square from the cliff areas. At the same time, it still chooses a safer path than Q-learning, as it avoids passing through the narrow passage and not going near the cliff areas.

N-STEP SARSA ALGORITHM

The n-step SARSA algorithm is an extension of SARSA. In this algorithm, the difference is that the agent also considers n number of future rewards and actions and then updates the Q-values. The equations that accomplish this added feature are described below:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

$$Q(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q(S_t, A_t)\right]$$

The first equation calculates the n-step return $G_{t:t+n}$, where we sum the rewards from R_{t+1} to R_{t+n}, each multiplied by the discount factor γ raised to the current step, and $Q_{t+n-1}(S_{t+n}, A_{t+n})$ is the

Figure 7: Greedy paths chosen by the Expected SARSA agent. Arrows indicate the optimal action at each state, with blue arrows showing the greedy path from both starting positions. The agent effectively avoids the cliff states and successfully reaches the goal, giving good policy performance.

Q-value of the state-action pair at n next steps, which is set to 0 if the episode terminates before n steps. The second equation updates the Q-value for the state S_t and action A_t, calculated by how much the current Q-value $Q(S_t, A_t)$ differs from the observed n-step return $G_{t:t+n}$.

To evaluate the n-step SARSA algorithm, we conducted an experiment testing the n values 1, 2, 5, 10, and 25, for the same number of repetitions and episodes as all the other experiments. We used a learning rate of $\alpha = 0.5$ as it had the best performance on the SARSA algorithm. As shown in Figure 8 all the learning curves show similar overall performance and converge to about the same average rewards. In addition, a small observation is that variance slightly increases as the n value increases.

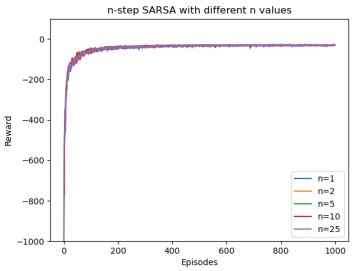

Figure 8: n-step SARSA performance over 100 repetitions of 1000 episodes for different n values, using $\alpha = 0.5$, $\epsilon = 0.1$, and $\gamma = 1.0$. The n values appear to converge to equal average rewards.

The fact that the average rewards for all different n are approximately equal is possibly due to the small state-action space which allows all n values to learn the optimal path. Generally, incorporating the multi-step updates supports considering several future rewards, which can assist in faster learning. As the n value increases the bias decreases, because we rely less on estimates, but the variance increases as the agent spends more time exploring inefficient paths as well. If we decide to set n to approximate infinity, the agent will wait until the end of the episode to update its Q-values. This is similar to the Monte Carlo Methods that give unbiased updates based on the actual results and not the estimates but with high variance and slower learning.

COMPARISON & CONCLUSION

To compare the performance of all four reinforcement learning algorithms analyzed in this report, we plotted the average rewards of their best configurations for each. The results are demonstrated in Figure 9. We see that all converge toward similar reward values, but their behavior differs.

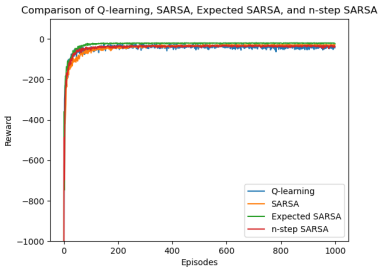

Figure 9: Comparing the performance of Q-learning with $\alpha = 0.5$, SARSA with $\alpha = 0.5$, Expected SARSA with $\alpha = 0.9$, and n-step SARSA with $n = 2$ and $\alpha = 0.5$, over 100 repetitions of 1000 episodes.

Among the algorithms that use 1-step updates, Expected SARSA has significantly the best overall performance as it learns quickly and shows minimal variance. This aligns with the performance shown in the greedy paths as it takes a relatively short path avoiding getting close to the cliffs, which combines efficiency and caution. On the other hand, Q-learning learns fast but has a high variance, mostly at the start of the episodes. This is reflected in its greedy paths that pass next to the cliffs to reach the goal as fast as possible, which are optimal but risky. Lastly, SARSA shows the slowest learning but keeps a stable performance with low variance, which matches the greedy paths that choose a safe route and thus have fewer penalties.

When we also consider the multi-step n-step SARSA algorithm, we see that its learning curve is stable and quick. In comparison with the other algorithms, we notice that its behavior is better than Q-learning and SARSA, but worse in speed than Expected SARSA, which is explained by its delayed updates when considering the future sequences of rewards.

In conclusion, model-free RL algorithms generally differ in how they operate. Higher learning rates show better performance than lower ones, with an exception in SARSA. Depending on the task we should choose different algorithms as they have different behavior traits. For example, sometimes the task expects us to be more cautious with the drawback being the lower learning speed. When considering a stochastic environment, algorithms differ from the first environment. We believe that for the next step, we should test these algorithms on a larger and more difficult environment, especially for n-step SARSA. Moreover, we could try a different action selection approach instead of ϵ-greedy to analyze how the behavior of the algorithms changes.

REFERENCES

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