INTRODUCTION TO RL ASSIGNMENT 2: MODEL-FREE REINFORCEMENT LEARNING

Bas Blokzijl (s2605457) and Enrico Bonsu (s3045072)

Universiteit Leiden
Introduction to Reinforcement Learning 2022-2023

1 INTRODUCTION

In this report, we take a look at three different reinforcement learning algorithms by applying them to a simple path-finding problem. The algorithms considered are Q-learning, SARSA, and Expected-SARSA. The environment consists of a grid of tiles where for each tile the agent has the ability to perform one out of four possible actions. These are moving up, down, left, or right. Furthermore, the environment contains two tiles that are used as the possible starting position and one tile that is used as the goal. Some of the tiles contain *cliffs* which, in case the agent steps on them, result in a penalty (large negative reward) for the agent and teleport the agent back to one of the starting positions. The goal of the agent is to find a route from starting position to the goal position.

1.1 THE ENVIRONMENT

The in the experiment we look at two different grid-based environments: Shortcut and WindyShortcut. Both have 12 rows and 12 columns, and the agent starts in one of two starting positions (chosen at random). The agent can move up, down, left, or right and receives a reward of -1 for each move that does not result in falling from a cliff or reaching the goal. There is one goal and several cliffs that results a penalty of -100 and teleport the agent to one of the starting positions (chosen at random). In WindyShortcut environment there is a 50% chance of the agent getting pushed one cell down after an action is taken.

2 Q-LEARNING

Q-learning is a reinforcement learning algorithm that uses state-action pairs to estimate the reward received for executing an action for a given state. These estimations are stored in a Q-table that stores the quality (estimate reward) of actions in a particular state. Q-learning is an off-policy algorithm. This means that there are two policies. The first policy b is called the behaviour policy, which generates data that is used to optimize the second policy π called the target policy. After each action taken the algorithm updates the state-action pair in the Q-table based on the rewards received by the agent.

2.1 METHODOLOGY

In our implementation of Q-Learning we generate data by using a $\epsilon-$greedy policy, while our target policy is using a greedy policy.
Definition *Let A be a set of actions and let S be the set of possible states. Given a Q table: $Q : S \times A$ and current state S_t, we define the ϵ-greedy policy as:*

$$\pi(s;\epsilon) : S \times A \to A : s \mapsto \begin{cases} \arg\max_{a \in \mathbb{A}} Q(S_t, a), & \text{with } \mathcal{P} = 1 - \epsilon \\ a \sim U(A), & \text{with } \mathcal{P} = \epsilon \end{cases} \tag{1}$$

Using our previous definition, we can now provide the $Q-$Learning algorithm.

Algorithm 1 Q-Learnig algorithm (off-policy TD control) for estimating target policy $\pi \approx \pi_*$ (1)

procedure
  Input: Exploration parameter $\epsilon \in [0, 1]$, learning rate $\alpha \in (0, 1]$, discount factor $\gamma \in \mathbb{R}^+$ let $\mathcal{E}$ be the set of episodes.
  Initialization: Initialize $\forall a \in \mathbb{A} \forall s \in S : Q(s, a)$ arbitrarily except that $Q(terminal, \bullet) = 0$. Let $\mathcal{M}$ be the set of possible starting positions.
  for each episode $\in \mathcal{E}$ **do**
    Initialize S.
    Set start $\sim U(\mathcal{M})$
    for each step of episode **do**
      Determine a_t ϵ-greedy using $S_t : a_t = \pi(S_t\ ;\ \epsilon)$
      Obtain reward: $r_t \sim p(r|a_t)$
      Get the next state from the environment using selected action: $S_{t+1} \leftarrow \texttt{env}(S_t)$
      $Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha * [r_t + \gamma \cdot \arg\max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, a_t)]$
      break for when S_{t+1} is terminal.
    end for
  end for
end procedure

2.2 The effect of learning rate α

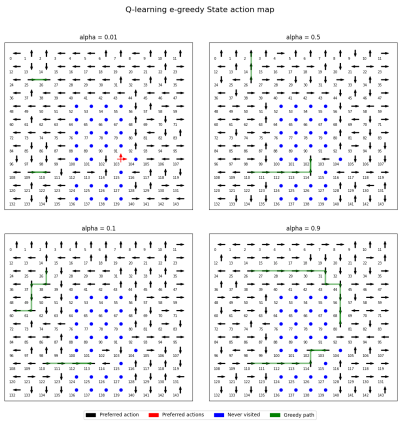

Figure 1: Q-learning with behaviour policy ϵ-greedy ($\epsilon = 0.1$) average performance for different values for α with episode time step size 1000 (100 repetitions).

In Figure 1 we first note that the learning rate α determines the extent to which newly acquired information overrides old information. We see that for low values of alpha, in the ShortCut environment, the agent performs rather poorly. It does manage to avoid cliffs, as shown by the greedy-outlined paths but it fails to find the path towards the goal. We expected to see this behaviour for low values of alpha as then the new information does not strongly alternate the values stored in the Q table, in turn leading to sub-optimal results. We find it interesting to see that even at low α values the agent still manages to avoid cliffs for the most part the large negative reward from stepping on cliffs pays off here.

3 SARSA

Just like the Q-learning algorithm the SARSA algorithm is a Temporal-Difference policy control. However, unlike the Q-learning algorithm, SARSA is an on-policy algorithm. This means that the SARSA algorithm attempts to evaluate or improve the policy that is used to make decisions. The algorithm is set to converge to the optimal policy given that all state-action pairs are visited an infinite number of times, and the policy converges in the limit to the greedy policy. The methodology is given next.

3.1 METHODOLOGY

Algorithm 2 SARSA (on-policy TD control) for estimating $Q \approx q_*$ (1)

procedure
  Input: small $\epsilon > 0$, step size $\alpha \in (0, 1]$, discount factor $\gamma \in \mathbb{R}^+$, let $\mathcal{E}$ be the set of episodes.
  Initialization: Initialize $Q(s, a), \forall s \in \mathbb{S}^+, a \in A(s)$, arbitrarily except that $Q(terminal, \bullet) = 0$. Let $\mathcal{M}$ be the set of possible starting positions.
  for each episode $\in \mathcal{E}$ **do**
    Initialize S
    Set start $\sim U(\mathcal{M})$
    Choose A from S using policy derived from Q
    for each step of episode **do**
      Take action A, observe R, S'
      Choose A' from S' using policy derived from Q
      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
      $S \leftarrow S'; A \leftarrow A'$
    end for
  end for
end procedure

3.2 THE EFFECT OF LEARNING RATE α

For our experiment we used a $\epsilon-$greedy policy (with ϵ = 0.1) to pick an action for the current state and the next state. We set the initial values for all state-action pairs $Q(s, a)$ to 0 and γ to 1. In the experiment we look at different values for α and look at how the α value influences the result. Out of all the α values only $\alpha = 0.9$ comes close to the goal. The biggest difference created by the increase in α value is that the policy is forcing itself leave the edge of the cliff. This is not surprising given that the we when we update our expected reward for a state-action pair we consider the expected reward for an action taken in the next state. The selection of the action is based on the policy (in our case ϵ-greedy). With our current set up we have a 10% chance of selecting a random action for our next state. When there are enough time steps for the episode, the effect of picking a random action close to the edge quickly evaluates the state to a bad state to be in. This effect is learned more quickly when the α value is large (see Figure 2).

4 WINDY ENVIRONMENT

The seen results are both performed in the Shortcut environment, which is deterministic. In this section we look at the performance of the algorithms using the Windy environment. This environment has a 50% of pushing the agent downward after performing an action.

4.1 RESULT

We observe from this experiment that both agents try to force a movement up, which could be the result of trying to cancel out the effect of the wind. Second (and more importantly) we observe that both agents avoid getting above the cliffs. This not surprising, given the large probability of getting pushed downward. Moving above the cliff would thus be a risk to get rewarded a penalty (when falling of the cliff). Both agents therefore are unable to come close to the goal (see Figure 4).

Figure 2: SARSA with $\epsilon-$greedy policy $\epsilon = 0.1$ average performance for different values for α with episode timestep size 1000 (100 repetitions).

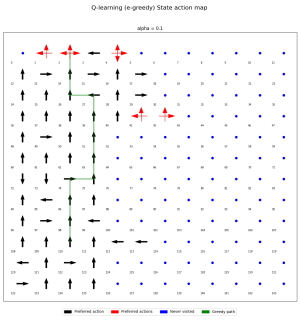

Figure 3: Q-learning with behaviour policy ϵ-greedy ($\epsilon = 0.1$) average performance with episode time step size 10000 (1 run).

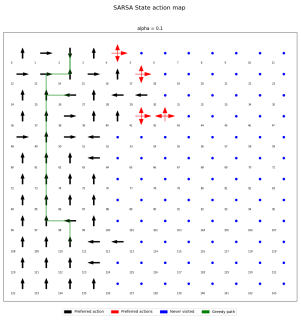

Figure 4: SARSA with $\epsilon-$greedy policy ($\epsilon = 0.1$) average performance with episode time step size 10000 (1 run).

5 Expected SARSA

Expected SARSA is a variant of the SARSA algorithm that estimates the Q-values using the expected value of the next state-action pairs. For our experiment Expected SARSA is an on-policy algorithm, however Expected SARSA can use a different policy to generate behaviour that is different from its target policy. This allows Expected SARSA to be an off-policy control algorithm.

5.1 Methodology

Before defining our algorithm we first need to define the following probability density function.
Definition $\mathrm{P}(S, a)$.
Let S_t be the current state and let S be the set of states. Given exploration parameter $\epsilon \in [0, 1]$ and $\mathcal{A}$ as the set of possible actions. For an action $a \in \mathcal{A}$ we define $\mathbb{P}(S_t, a)$ as follows:

$$\mathbb{P}(S_t, a) : S \times \mathcal{A} \to [0, 1] : a \mapsto \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & a = \arg\max_{a \in \mathbb{A}} Q(S_t, a) \\ \frac{\epsilon}{|\mathcal{A}|} & \textit{otherwise} \end{cases} \tag{2}$$

Algorithm 3 Expected SARSA (on-policy TD control) for estimating $Q \approx q_*$ (1)

procedure
  Input: Exploration parameter $\epsilon \in [0, 1]$, learning rate $\alpha \in (0, 1]$, discount factor $\gamma \in \mathbb{R}^+$ let $\mathcal{E}$ be the set of episodes.
  Initialization: Initialize $\forall a \in \mathbb{A} \forall s \in S : Q(s, a)$ arbitrarily except that $Q(terminal, \bullet) = 0$. Let $\mathcal{M}$ be the set of possible starting positions.
  for each episode $\in \mathcal{E}$ **do**
    Initialize S
    Set start $\sim U(\mathcal{M})$
    Choose A from S using policy derived from Q
    for each step of episode **do**
      Determine a_t ϵ-greedy using $S_t : a_t = \pi(S_t\ ;\ \epsilon)$
      Obtain reward: $r_t \sim p(r|a_t)$
      Calculated expected reward: $\hat{r}_{t+1} = \sum_{a \in A} Q(S_{t+1}, a) \cdot \mathbb{P}(S_{t+1}, a)$
      Get the next state from the environment using selected action: $S_{t+1} \leftarrow \texttt{env}(S_t)$
      $Q(S, a) \leftarrow$Q(S,a) + $\alpha \cdot [r_t + \gamma \cdot \hat{r}_{t+1} - Q(S_t, a)]$
      break for when S_{t+1} is terminal.
    end for
  end for
end procedure

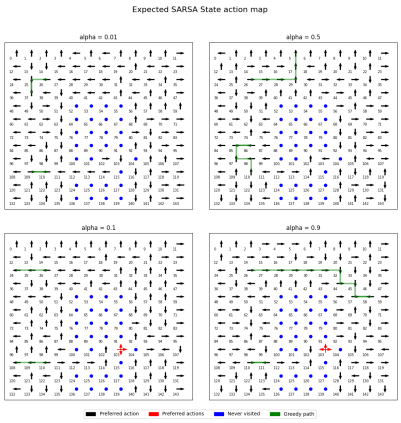

Figure 5: Expected SARSA with $\epsilon-$greedy policy $\epsilon = 0.1$ average performance for different values for α with episode time step size 1000 (100 repetitions).

5.2 The effect of learning rate α

Because Expected SARSA calculates the expected reward of a state to be the weighted expected reward over all the actions in that state, we observed that increasing the value for α results in the agent above coming close to the cliffs. This makes perfectly sense, since the expected reward for states close to the edge also considers the penalty for falling of the cliff from that state. Given that the penalty is large, the agent therefore always expect the states close to cliffs to be a bad state to be at. For lower α values this effect is hard to observe, since updating the state-action pair is much slower, thus more episodes are needed to generate the same affects as seen with larger α values (see Figure 5).

6 Bonus: SARSA with $\epsilon - soft$ policy ($\epsilon = \frac{1}{t}$)

In this section we look at an other implementation of the SARSA control policy. In our initial implementation, we used the $\epsilon-$greedy with $\epsilon = 0.1$. For this implementation we set ϵ to $\frac{1}{t}$, where t is the time step of the episode (with $t = 1, 2, 3, ..$).

6.1 RESULT

We are interested in the performance of this variant on our experiment compared to our original implementation. Given that SARSA is an on-policy control policy we know that SARSA with $\epsilon-$greedy will converge, however this is will not generate the optimal policy. This is because the randomness generated of ϵ will persist in the expected reward for the state-action pairs. This problem can be solved if we let ϵ converge in the limit to an greedy policy (1). This is the case for $\epsilon = \frac{1}{t}$. Comparing the results of this version (Figure 6) with the experiment results of the original implementation of SARSA (Figure 2) we observe a significant improvement for larger values α. For α values 0.5 and 0.9 the policy would lead us confidently through the narrow path to the goal (when visiting the bottom starting position). The other starting position also shows very strong improvement. Not only is the greedy path from start point very close to the goal it is also that states close to the cliff now progress towards the goal. From this we conclude that this version of the SARSA implementation performs better in the Shortcut environment for larger α values, than the original implementation.

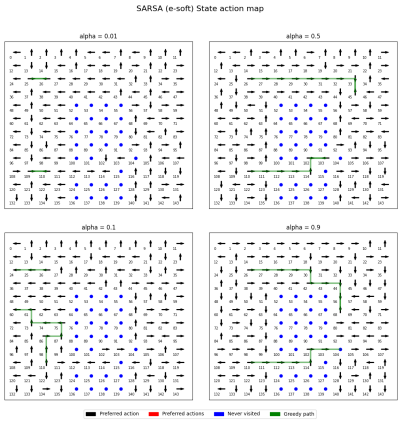

Figure 6: SARSA with $\epsilon - soft$ policy $\epsilon = \frac{1}{t}$ average performance with episode timestep size 1000 (100 repetitions).

7 CONCLUSION

In general for our experiment having a large value for learning rate α improves the result significantly. For the first three algorithms discussed the Q-learning algorithm out performance both SARSA and Expected SARSA. The reason for this result is that both SARSA and Expected SARSA avoid coming close to the edge of an cliff. Given that our environment is deterministic this behaviour is sub optimal. The result of our SARSA implementation with the $\epsilon-$soft policy $\epsilon = \frac{1}{t}$ however is the performing algorithm. The main reason for this is that exploring this environment does not require time steps. Thus switching from an all explore to an all exploit approach for selecting actions results to a nearly optimal policy within 1000 time steps. Considering the algorithms look at in this report we conclude that model-free RL algorithms can be very useful for finding an optimal policy, given that enough episodes (with enough time steps) can be generated. Also allowing us only to consider the states and actions results quick overview of the preferred action for each state.

8 BIBLIOGRAPHY

REFERENCES

[1] Sutton, R. S., Barto, A. G. (2020). Reinforcement Learning: An Introduction 2nd edition. The MIT Press.