EL2805: REINFORCEMENT LEARNING
LAB1

Author
Adel Abdelsamed 2000321-T455
Alexander Weers 19990711-T455

Contents

1 Problem1: The Maze and the Random Minotaur

1.1 Task A

The MDP is characterized by the tuple $\{T, \mathcal{S}, \mathcal{A}, (\mathcal{A}_s, p_t(\cdot|s,a), r_t(s,a), 1 \leq t \leq T, s \in S, a \in A_s)\}$.
First, the state space $\mathcal{S}$ is defined. We adopt the same coordinate system as in Lab 0. Let $\mathcal{S}_o$ define the set of (x,y)-positions in the grid, where there is an obstacle

$$\mathcal{S}_o = \{(0,2), (1,2), (2,2), (3,2), ...\}.$$

Furthermore, let $\mathcal{S}_{grid}$ be the set of all (x,y)-positions in the grid

$$\mathcal{S}_{grid} = \{0, 1, ...7\} \times \{0, 1, ...6\}$$

Hence, we define the state space

$$\mathcal{S} = \{(p_x, p_y, m_x, m_y)|\ (p_x, p_y) \in \mathcal{S}_{grid} \setminus \mathcal{S}_o,\ (m_x, m_y) \in \mathcal{S}_{grid}\},$$

Since the minitaur can walk over the obstacles, its position is not restricted within the grid. Since we only have control over the player's position, we define the action space $\mathcal{A}$ as

$$\forall s : \mathcal{A}_s = \{up, down, left, right, stay\}.$$

Note, the admissible position for the player depends on the player's position at a given time. In our formulation we do not restrict the action space for any states, but deal with the walls and obstacles by defining the transition probabilities and the reward function appropriately.

In the following we describe the transition probabilities. The transition of the player is deterministic, while the minitaur follows a random walk and hence its movement is stochastic in nature. Hence the transition probabilities is stochastic and consists of the joint probabilities. Since the behaviour of the player is deterministic, the transition probabilities are characterized by the probabilities of the minitaur. At stage t and in state $s_t = (p_x, p_y, m_x, m_y)_t$ choosing action a_t will lead to $s' = s_{t+1} = (p_x, p_y, m_x, m_y)_{t+1}$, where the Manhattan distance $d_M(\cdot, \cdot)$ of the (x,y)-position of the player and the minitaur at stage t+1 can differ only by maximally one from the position at stage t. The transition probabilities are

1. If $(p_x, p_y)_{t+1} \notin \mathcal{S}_{grid}$, then $p_t(s'|s,a) = 0$.
2. If $(p_x, p_y)_{t+1} \in \mathcal{S}_{grid}$, then $p_t(s'|s,a) = \frac{1}{N_m}$, where N_m denotes the admissible directions of the minitaur at stage t.
3. If $(p_x, p_y)_t = (m_x, m_y)_t$, then $p_t(s'|s,a) = 1$, where s' = s.
4. If $(p_x, p_y)_t = (5, 6)_t$, then $p_t(s'|s,a) = 1$, where s' = s.

The last two conditions ensure that if the game is lost or won the state is the same for all future stages.

Moreover, the rewards function $r(s_t, a_t)$ abides by the following rules. At stage t and in state $s_t = (p_x, p_y, m_x, m_y)_t$ choosing action a_t will lead to $s_{t+1} = (p_x, p_y, m_x, m_y)_{t+1}$

1. If $(p_x, p_y)_{t+1} \in \mathcal{S}_o$, then $r(s_t, a_t) = -\infty$.
2. If $(p_x, p_y)_{t+1} = (m_x, m_y)_{t+1}$, then $r(s_t, a_t) = -\infty$.
3. If $(p_x, p_y)_t = (5, 6)$ and $(p_x, p_y)_{t+1} \neq (m_x, m_y)_t$, then $r(s_t, a_t) = 0$.
4. Otherwise $r(s_t, a_t) = -1$.

Since the rewards function is stochastic, the average reward $\mu(s, a)$ is used. This can be computed by considering all the next states given a state and action and weighting the rewards received from the cases above with the probability of moving to this next state. It is noted that the modelled reward function and the transition probabilities are stationary. Lastly, since the problem is modelled as a finite-horizon MDP, the horizon is specified as T.

1.2 Task B

Again the MDP is characterized by the tuple $\{T, \mathcal{S}, \mathcal{A}, (\mathcal{A}_s, p_t(\cdot|s, a), r_t(s, a), 1 \leq t \leq T, s \in S, a \in A_s)\}$. Now, the the player and the minitaur take turns playing. The state space $\mathcal{S}$ is defined as

$$\mathcal{S} = \{(p_x, p_y, m_x, m_y, turn)|\ (p_x, p_y) \in \mathcal{S}_{grid} \setminus \mathcal{S}_o,\ (m_x, m_y) \in \mathcal{S}_{grid}, turn \in \{p, m\}\}.$$

The extended state contains the variable turn which stores who played last.

If at stage t in state $s_t = (p_x, p_y, m_x, m_y, turn)_t$ we define the following sets of admissible actions.

1. If $turn_t$ = p, $\mathcal{A}_s = \{do\ nothing\}$.
2. If $turn_t$ = m, $\mathcal{A}_s$ is identical to Task A.

Since the players take turns, the deterministic behaviour of the player is decoupled from the stochastic behaviour of the minitaur. At stage t and in state $s_t = (p_x, p_y, m_x, m_y, turn)_t$ choosing action a_t will lead to $s_{t+1} = (p_x, p_y, m_x, m_y, turn)_{t+1}$, where again the Manhattan distance $d_M(\cdot, \cdot)$ between stage t+1 and t for the player and the minitaur must not exceed one. The transition probabilities if $turn_t = m$ are

1. If $(p_x, p_y)_{t+1} \in S_{grid} \setminus S_o$, then $p_t(s'|s, a) = 1$.
2. If $(p_x, p_y)_t = (m_x, m_y)_t$, then $p_t(s'|s, a) = 1$, where s' = s.
3. If $(p_x, p_y)_t = (5, 6)_t$, then $p_t(s'|s, a) = 1$, where s' = s.

The next state in all cases is $s' = (\cdot, \cdot, \cdot, \cdot, p)_{t+1}$.

If $turn_t = p$, then, The transition probabilities are the same as in Task A. The next state is then $s' = (\cdot, \cdot, \cdot, \cdot, m)_{t+1}$.

Moreover, the rewards function $r(s_t, a_t)$ abides by the following rules. At stage t and in state $s_t = (p_x, p_y, m_x, m_y, turn)_t$ choosing action a_t will lead to $s_{t+1} = (p_x, p_y, m_x, m_y, turn)_{t+1}$. We differentiate between $turn_t$ = p and m. For $turn_t = p$, we define

1. If $(p_x, p_y)_{t+1} = (m_x, m_y)_{t+1}$, then $r(s_t, a_t) = -\infty$.
2. Otherwise $r(s_t, a_t) = 0$.

$turn_t = m$ we define the same rewards as in Task A. Again since, the rewards function is stochastic, the average reward $\mu(s, a)$ is used. This can be computed by considering all the next states given a state and action and weighting the rewards received from the cases above with the probability of moving to this next state. Lastly, since the problem is modelled as a finite-horizon MDP, the horizon is specified as T.

Technically, there is a difference from Task A. If the minoutaur and the player are in neighbouring grids, then the probability of the minotaur eating the player in the next state is zero in Task A, since they both move in the same time and the minitaur can't stay in the same position. This is not true in Task B anymore, since the player stays in the same position while the minitaur is playing. Hence, the minitaur is more likely to catch the player if they are taking turns playing.

1.3 Task C

The optimal policy is to take the shortest path to the exit while avoiding the minitaur. The optimal policy is shown in Figure 1

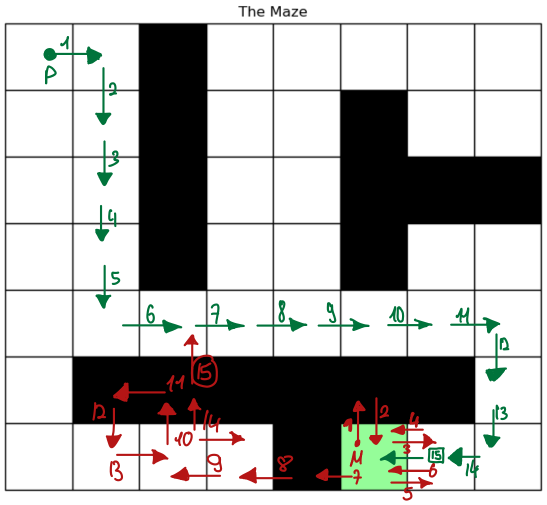


Figure 1: Illustration of a sample episode. Player's moves are shown in green, minitaur's in red.

1.4 Task D

Yes, there is a difference. If the minitaur is allowed to stay in the same position, then there is a positive probability that the minitaur blocks key grids leading to the exit and hence prohibiting the player to reach the exit. Hence, we should expect the probability of getting out successfully to be lower in the case the minitaur can stay in the same grid. This is verified by plotting the probability of getting out successfully for different horizons in the cases the minitaur can stay in the same position and in the case it is not allowed. This is illustrated in Figure 2. Clearly, the probability of exiting the maze successfully is zero, whenever the time-horizon is smaller than 15, which is the minimal steps required to exit the maze.

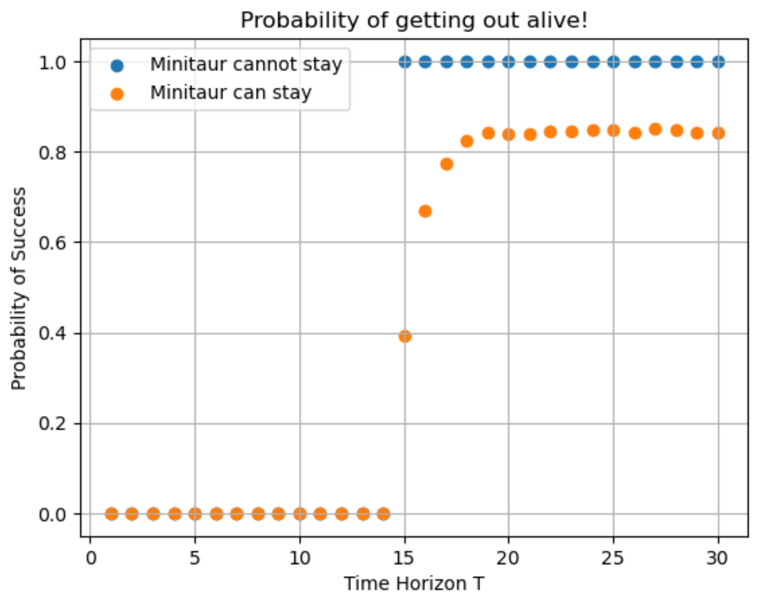


Figure 2: Probability of exiting the maze successfully with the minitaur allowed to stay in the same position and without.

1.5 Task E

Now, the problem is modelled as an infinite-horizon MDP. Since the reward function and transition probabilities defined in Task A are stationary, we adopt the same state space, action space, reward function and transition probabilities. It remains to specify the discount factor.

Furthermore, now we assume the player is poisoned and that the player's life is geometrically distributed with mean 30. We make use of the discount factor to account for the random time horizon as discussed in Lecture 02. In this case the time horizon is a random variable T that is geometrically distributed $T \sim geom(1-\lambda)$. The discount factor can be computed using the expression for the mean of T.

$$\mathbb{E}[T] = \frac{1}{1-\lambda}$$

resulting in

$$\lambda = 1 - \frac{1}{\mathbb{E}[T]} = \frac{29}{30}.$$

1.6 Task F

The probability of getting out alive was computed by simulating 10000 games. In the case the agent gets out we compute the probability that $P(T \leq t_{out})$ which is obtained using the cumulative distribution of the geometric distribution $T \sim geom(1-\lambda)$. We obtain the probability 0.8332 for successfully exiting the maze.

1.7 Task G

Task 1: In an off-policy learning method, the learner learns the value of the optimal policy and not the value of the behaviour policy. In on-policy methods the agent learns the value of the policy that is being used to generate the data.
Task 2: In Q learning the following conditions must be met for convergence:

- Step size (α_t) must be square summable but not summable, i.e. $\sum_t \alpha_t = \infty,\ \sum_t \alpha_t^2 < \infty$
- Behaviour policy π_b must visit every (state, action) pairs infinitely often.

In SARSA with ϵ-greedy learning the following conditions must be met for convergence:

- Step size (α_t) must be square summable but not summable, i.e. $\sum_t \alpha_t = \infty,\ \sum_t \alpha_t^2 < \infty$
- The policy π_t is ϵ-greedy policy w.r.t $Q^{(t)}$.

1.8 Task H

The problem can be modelled as a discounted infinite-horizon MDP as in Task E. However, there are some differences compared to the MDP in Task E.

Firstly, the player's life is now geometrically distributed with mean 50. This will lead to a changed discount factor $\lambda = \frac{49}{50}$.

Secondly, the player must now reach C to get the keys in order to exit the maze. Hence, the state tuple will be extended to an additional variable that stores if the player has the keys, i.e if the player visited C. The new state space $\mathcal{S}$ is defined as

$$\mathcal{S} = \{(p_x, p_y, m_x, m_y, keys)|\ (p_x, p_y) \in \mathcal{S}_{grid} \setminus \mathcal{S}_o,\ (m_x, m_y) \in \mathcal{S}_{grid}, keys \in \{yes, no\}\}.$$

The action space for all states remains the same

$$\forall s : \mathcal{A}_s = \{up, down, left, right, stay\}.$$

The minitaur now moves with probability 35% towards the player and with the probability 65% uniformly at random in all directions. Still the minitaur can't remain in the same position.

If at stage t and in state $s_t = (p_x, p_y, m_x, m_y, key)_t$ choosing action a_t will lead to $s_{t+1} = (p_x, p_y, m_x, m_y, key)_{t+1}$, where again the Manhattan distance between stage t+1 and t for the player must remain one and the minitaur's can be zero or one. The transition probabilities are

1. If $(p_x, p_y)_{t+1} \notin \mathcal{S}_{grid}$, then $p_t(s'|s, a) = 0$.
2. If $(p_x, p_y)_{t+1} \in \mathcal{S}_{grid}$, then $p_t(s'|s, a) = \frac{1}{N_m} \cdot \frac{13}{20}$, where N_m denotes the admissible directions of the minitaur at stage t. If $(p_x, p_y)_{t+1} = C$, then $(keys)_{t+1} = 1$.
3. If $(p_x, p_y)_{t+1} \in \mathcal{S}_{grid}$, then $p_t(s'|s, a) = \frac{1}{N_c} \cdot \frac{7}{20}$, where N_c denotes the admissible directions of the minitaur at stage t that will move the minitaur closer to the player in the next step. If $(p_x, p_y)_{t+1} = C$, then $(keys)_{t+1} = 1$.

4. If $(p_x, p_y)_t = (m_x, m_y)_t$, then $p_t(s'|s, a) = 1$, where s' = s.

5. If $(p_x, p_y)_t = (5, 6)_t$ and $(keys)_t = 1$, then $p_t(s'|s, a) = 1$, where s' = s.

At stage t and in state $s_t = (p_x, p_y, m_x, m_y, keys)_t$ choosing action a_t will lead to $s_{t+1} = (p_x, p_y, m_x, m_y, keys)_{t+1}$. The rewards function is defined as

1. If $(p_x, p_y)_{t+1} \in \mathcal{S}_o$, then $r(s_t, a_t) = -\infty$.

2. If $(p_x, p_y)_{t+1} = (m_x, m_y)_{t+1}$, then $r(s_t, a_t) = -\infty$.

3. If $(p_x, p_y)_t = (5, 6)$ and $(p_x, p_y)_{t+1} \neq (m_x, m_y)_t$ and $(keys)_t = 1$, then $r(s_t, a_t) = 0$.

4. Otherwise $r(s_t, a_t) = -1$.

Since the rewards function is stochastic, the average reward $\mu(s, a)$ is used. This can be computed by considering all the next states given a state and action and weighting the rewards received from the cases above with the probability of moving to this next state.

1.9 Bonus Task I

In Task I, the MDP defined in Task H is solved using an online version of Q-learning. The behaviour policy is chosen to be epsilon-greedy w.r.t to the current estimate of the Q-function $Q_\pi^{(t)}$. The pseudocode for the implementation of Q-learning is provided below.

Listing 1: Q-learning implementation pseudocode

```
Given: no_episodes, epsilon, stepsize_exp
        # Initialization:
        # The variables involved in the Q learning
        Q       = np.zeros((n_states, n_actions));
        counter = np.zeros((n_states, n_actions));

        # Iterate through episodes:
        for epsiode_no in range(no_episodes):

            # Initialize environment and start state
            s = env.map[start];

            while not episode_ended:
                # Select action a_t based on an epsilon
                   greedy-policy
                action = env.e_greedy(s, Q,  epsilon)
                # Increase counter of visiting (s, a)
                counter[s, action] +=  1
                # Observe next state s_(t+1):
                next_s = env.get_next_state(s, action)

                # Compute the step size
```

```
                    alpha_t = 1/pow(counter[s, action],
                        stepsize_exp)
                    # Compute TD Target
                    y = r[s,action] + gamma*np.max(Q[next_s,:])
                    # Update of the Q values
                    Q[s,action] += alpha_t*(y - Q[s,action])

                    # Update the state
                    s = next_s;

                    # Check if episode has ended?
                    if (episoded_ended):
                        break;

          # From Q construct the policy
          policy = np.argmax(Q,1)
```

Given a number of episodes, the Q values and the counter for visiting every (state-action) pair is initialized to zero. Moreover, in an iteration through all episodes the environment is reset by starting from the initial state. The following steps are repeated until the episode has terminated.

- Action a_t is selected using an ϵ-greedy policy w.r.t to the current estimate of Q.
- Counter of visiting (s_t, a_t) is incremented.
- Based on (s_t, a_t) the next stepp $s' = s_{t+1}$ is observed.
- Q-function is updated towards the computed TD-target using the equation

$$Q(s,a) = Q(s,a) + \frac{1}{n(s,a)^{\alpha}} \cdot (r(s,a) + \gamma \max_b Q(s',b) - Q(s,a)) \tag{1}$$

- State is updated by setting $s_t = s_{t+1}$

The following Figure 3 shows the convergence of the value of the initial state for two different exploration parameters ($\epsilon_1 = 0.25$, $\epsilon_2 = 0.75$) using a step-size of $\frac{1}{n(s,a)^{2/3}}$, where n(s,a) is the number of visits of the state action pair (s,a).

In order to obtain the optimal value of the initial state, value iteration was used. The initial state has an optimal value of -22.1691. Q-learning was run for 300000 episodes to reveal the full trend. Clearly, convergence is faster when $\epsilon = 0.25$ as it approaches the optimal value after approximately 150000 episodes. This result is sound, since an increase in the exploration parameter will lead to the agent spending more time exploring random actions. However, when ϵ is small, the agent exploits his estimates of the Q-function to generate actions.

Theoretically, the Q-learning algorithm will converge to the optimal Q-function independent of the initial values of the Q-function. However, using proper initialization affects the convergence speed highly. As we are dealing with a MDP that accumulates only negative rewards and the goal is distant, initializing the Q-function to zero constitutes an optimistic

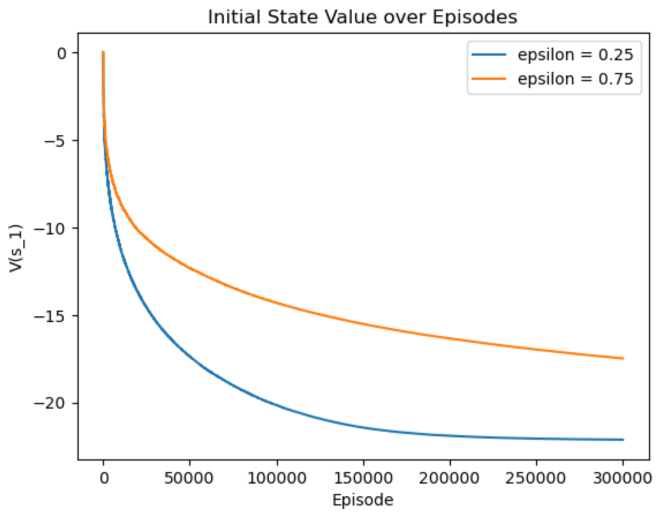


Figure 3: Q-learning convergence for two different exploration parameters: $\epsilon_1 = 0.25$ and $\epsilon_2 = 0.75$

initialization as described in Sutton's book. As no state can exceed zero rewards, the initial values of the Q-function incentivises exploration of state-action pairs that have not yet been tried.

Furthermore, if the norm of the initial Q-function is close to the optimal Q-function, convergence occurs in far less episodes. This was tested by initializing the Q-function by estimates from previous runs that has not fully converged. This means the choice of the initial Q-function affects convergence.

Lastly, we fix the exploration parameter $\epsilon = 0.2$ and illustrate the convergence for two different step sizes as seen in Figure 4. For $\alpha = 2/3$, Q-learning converges to the optimal

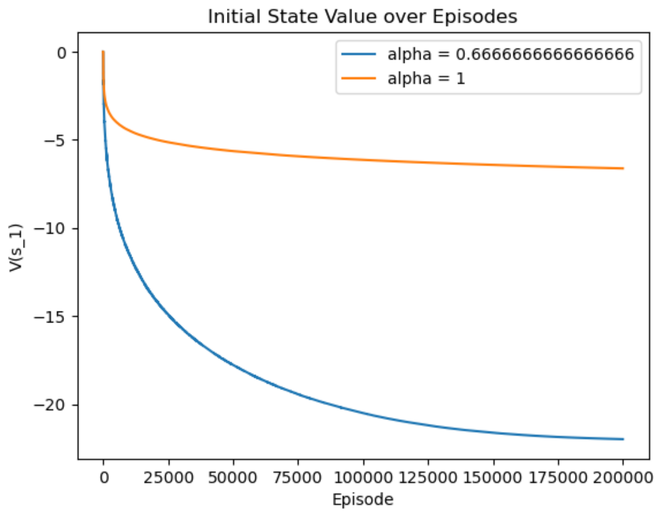


Figure 4: Q-learning convergence for two different step-size exponents: $\alpha_1 = 2/3$ and $\alpha_2 = 1$

value function. However, for $\alpha = 1$ the obtained Q-function is sub-optimal. Since an increase in α would result in $\frac{1}{n(s,a)^\alpha}$ going to zero faster, which in this case occurs before we converge to the optimal Q-values.

1.10 Bonus Task J

The difference to Q-learning is in the steps in each episode. In SARSA the following set of steps are done.

- Action a_t is selected using an ϵ-greedy policy w.r.t to the current estimate of Q.
- Counter of visiting (s_t, a_t) is incremented.
- Based on (s_t, a_t) the next step $s' = s_{t+1}$ is observed.
- Action $a' = a_{t+1}$ is selected using an ϵ-greedy policy w.r.t to the current estimate of Q.
- Q-function is updated towards the computed TD-target using the equation

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

- State is updated by setting $s_t = s_{t+1}$

Using a step-size law of $\frac{1}{n(s,a)^{2/3}}$ the SARSA algorithm faces challenges converging to a near optimal policy for the exploration rates $\epsilon_1 = 0.1$ and $\epsilon_2 = 0.2$ in 200K steps as shown in Figure 5. The player gets stuck in the initial blocks and never gets to exit the maze due to the distant goal. Convergence was seen to improve using other step-size rules and diminishing exploration rates.

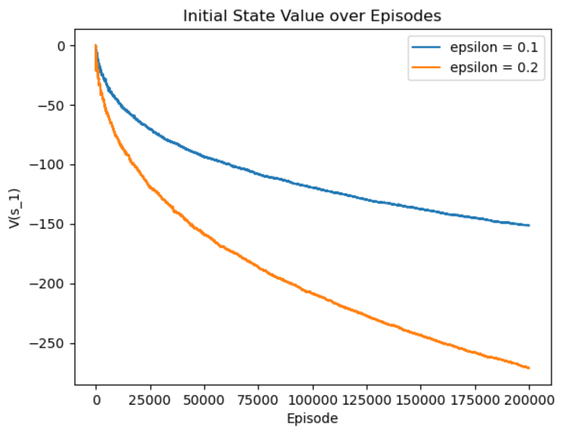


Figure 5: SARSA algorithm with a step-size rule $\frac{1}{n(s,a)^{2/3}}$ for two different exploration parameters $\epsilon_1 = 0.1$ and $\epsilon_2 = 0.2$.

Now we make use of the following adaptive step-size law $\epsilon_k = \frac{1}{k^\delta}$ with $\delta \in (0.5, 1]$. Here, the SARSA algorithm converges as shown in Figure 6. Using diminishing exploration rates allows the algorithm to converge to the true value of the initial state -22.1691.

If $\alpha > \delta$, then the step-size approaches zero quicker than the exploration rate. This result is not desirable, since the step-size should only diminish if the Q-function approaches its optimal values. This should occur when exploration has died out allowing the Q-function

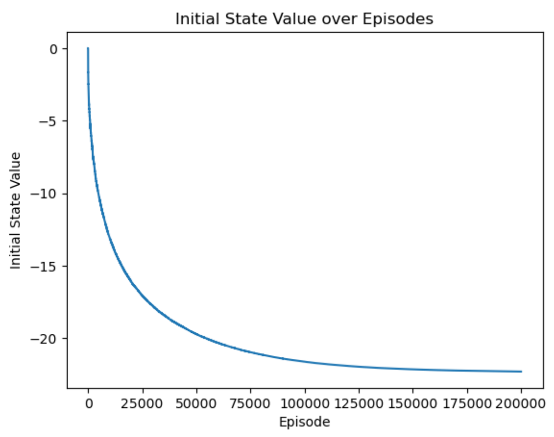


Figure 6: SARSA algorithm with diminishing exploration rate $\epsilon_k = \frac{1}{k}$ and step-size $\frac{1}{(1+n(s,a))^{2/3}}$

.

to improve by acting greedily w.r.t to the current estimate. Hence, it is better to choose $\alpha < \delta$.

1.11 Bonus Task K

The probability of getting out alive using the policies derived from Q-learning and SARSA are shown in Table 1.

Table 1: Probability of getting out of the maze successfully.

Q-learning	SARSA
0.46579	0.50987

2 Problem 2: RL with linear function approximators

2.1 3 (b): Implementation

Here we refer to the code.

2.2 3 (c): Training process

We implemented a SARSA(λ) Agent, that is able to learn a Q-function in the Mountain-Car environment, such that the running average reward over 50 epochs is above -135 (see 7).

2.2.1 SARSA(λ) Agent

The SARSA(λ) Agent is a linear function approximation of the Q-function using the Fourier Basis. The eligibility traces are intended to approximate the Forward(λ) algorithm, while allowing update rules at each step.

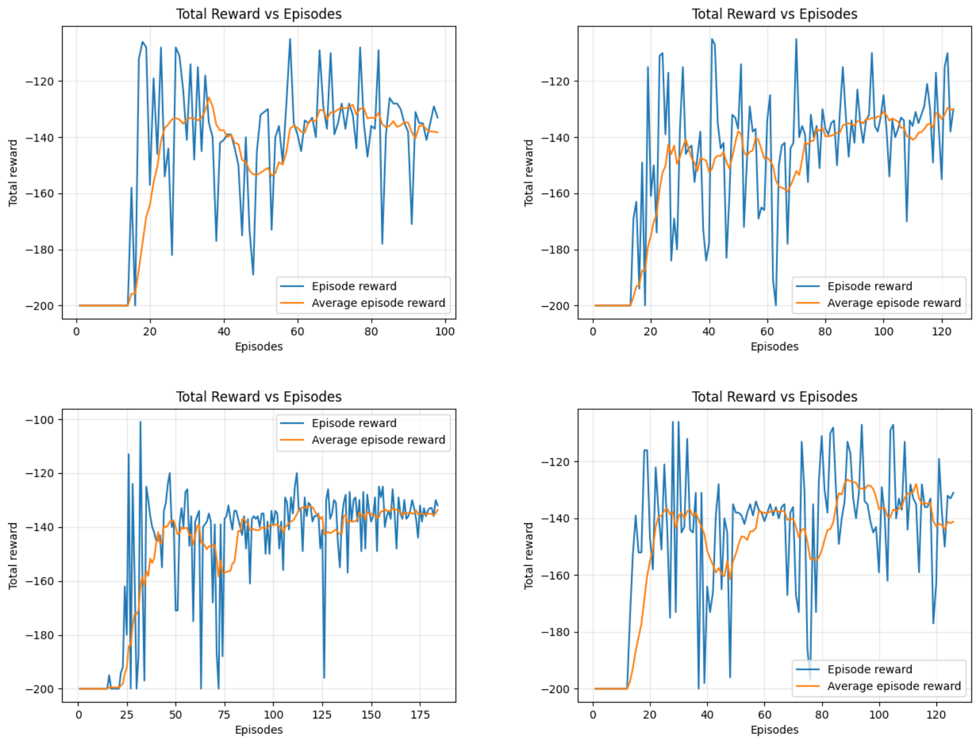


Figure 7: Four different training runs. The time to reach the goal (50-epoch average of higher than -135) depends on the random initialization and the ϵ-greedy policy.

We use a mixture of coupled and decoupled basis vectors:

$$\eta = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 2 & 0 \\ 2 & 1 \\ 0 & 2 \\ 1 & 2 \\ 2 & 2 \end{bmatrix}$$

where the row i corresponds to the basis vector η_i. Using this matrix η we can easily calculate the basis functions $\phi_i(s) = \cos(\pi \eta_i^T s)$. Initially, all combinations of basis vectors of order 2 were tried, leaving out the bias term, where $\eta_i = \begin{bmatrix} 0 & 0 \end{bmatrix}$
Besides a matrix of weights $\mathbf{w} \in \mathbb{R}^{k \times m}$, with k being the number of possible actions and m the feature size, the agent uses an eligibility trace $\mathbf{z} \in \mathbb{R}^{k \times m}$ during training to correctly update the actions that most contributed to the reward.
After initializing the values ($w_{ij} \sim \mathcal{U}_{[0,1]}$ and $z_{ij} = 0$), we first pick an action a_t in state s_t, which is chosen by a ϵ-greedy policy. After performing a_t we receive the reward r_t from the environment and reach s_{t+1}. We call the `update` method of the agent with s_t, a_t, r_t and s_{t+1} as arguments. This method first determines the next action a_{t+1} based on s_{t+1} and

then updates the values of $\mathbf{w}$ and $\mathbf{z}$ as described in the task:

$$\mathbf{z}_a \leftarrow \begin{cases} \gamma\lambda\mathbf{z}_a + \nabla_{w_a} Q_w(s_t, a) & \text{if } a = a_t \\ \gamma\lambda\mathbf{z}_a & \text{otherwise} \end{cases}$$

and $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta_t\mathbf{z}$ with $\delta_t = r_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)$. Note that $\nabla_{w_a} Q_w(s_t, a) = \nabla_{w_a} w_a^\top \phi(s_t) = \phi(s_t)$.
We train our `Agent` for a maximum number of 200 epochs, but stop as soon as the average reward is above -135. After each epoch the eligibility trace $\mathbf{z}$ is reset to zero values.
We used the clipping of egibility tree to keep those values between -5 and 5 and tested a few different values for λ, γ, ϵ and α. The final values can be found in Figure 2.
During each epoch the number of actions was limited to 200, which restricts the exploration possibilities per epoch.

Learning rate	$\alpha = 0.0015$
Greediness probability	$\epsilon = 0.001$
Feature size	$m = 8$
Eligibility trace	$\lambda = 0.9$
Discount factor	$\gamma = 1$

Table 2: Hyperparameter for SARSA agent

2.3 3(d): Analysis of policy

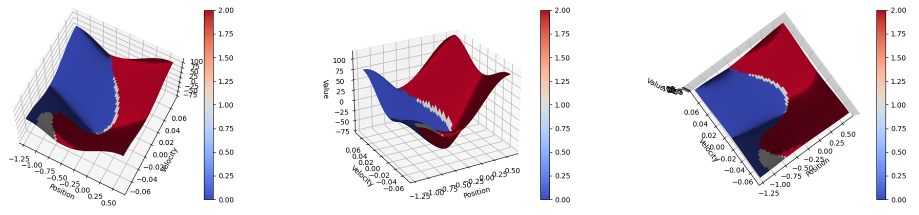


Figure 8: Three different perspectives on a plot of the value function and action (0: *Left*; 1: *Neutral*; 2: *Right*) with highest values for the state space (position, velocity).

Four plots of the total episodic reward across multiple epochs during training can be found in Figure 7. For the first ten epochs the agent is not able to reach the goal with 200 actions, but still acquires knowledge about the environment. After 10 to 20 epochs the agent consistently observes increased episodic rewards and only reaches the 200 actions limit a couple of times afterwards. After roughly 20 to 30 epochs the agent seems to understand the environment well enough and keeps the episodic reward for most epochs above -150. The 10-epoch average continues to rise as the outliers of very low reward get less frequent. In Figure 8 we show a plot of the values for state space and highlight the action with the highest value. Positions at the borders have higher values than positions near the middle, which makes sense intuitively. Since the car's motor is underpowered it should use the mountains to gain inertia thereby increasing its energy, ultimately reaching the goal at the top of the hill. For the velocity the value distribution is similar: high absolute velocities

have higher values than slow velocities, which follows the same argument. The chosen action mainly depends on the position rather than the velocity. Closer to the goal the agent tries to push over the hill, while the agent uses the left hill to increase its potential energy as mentioned before.
As described in 2.2.1 we initially did not include $\eta_i = \begin{bmatrix} 0 & 0 \end{bmatrix}$ in η. Doing so does not lead to a noticeable difference in convergence speed or changes in the value/action plots:

$$\begin{aligned}\phi_0 &= \cos(\pi \eta_0^\top s)\\ &= \cos(\pi[0,0]s)\\ &= \cos(0) = 1\end{aligned}$$

For $\eta_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$ the Fourier basis function returns a constant value not depending on the state. This leads to w_{a0} acting as a bias, a constant offset for this action a. This could prefer some actions over others independent of the current state. This can also be verified by observing the state space equations for the mountain car.

$$\begin{aligned}x_1^{k+1} &= x_1^k + x_2^k\\ x_2^{k+1} &= -0.0025\cos(3x_1^k) + x_2^k + 0.001u^k\end{aligned}$$

Clearly, the input is not decoupled from the system and hence a constant offset does not increase the rewards.

2.4 3(e): Analysis of training process

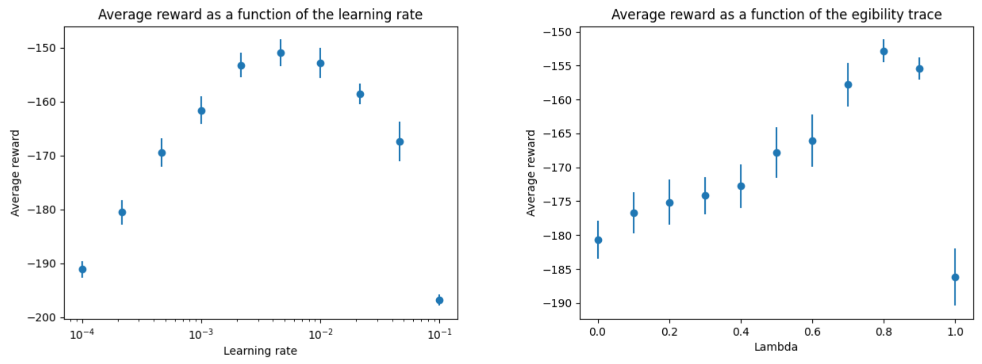


Figure 9: Plot of total average reward and 95% confidence intervals for different learning rates α (left) and egibility traces λ (right).

2.4.1 (1): Average total reward as a function of hyperparameter

To further analyze the importance of hyperparameter choices, we run some experiments for different values. All experiments in this section are run for 50 trials per value and we report the average episodic reward as well as the 95% confidence interval of it.
We choose a logarithmic scaled distribution for the learning rates α to better scrutinize a wide range of different values. In 9 (left) it is shown that a learning rate $\alpha = 1 \cdot 10^{-2}$

is optimal, keeping the other hyperparameters fixed. Neither incresing nor decreasing the learning rate provides a benefit. This analysis shows, that we achieved a close-to-optimal value for the learning rate by manually optimizing the hyperparameters.
Similarly we examine the influence of the egibility trace λ on the average total reward (see. 9 (right)). High values for λ seem to perform better and our initial choice of $\lambda = 0.9$ seems to be a good value, although $\lambda = 0.8$ could perform better.

2.4.2 (2): Initialization strategies for Q-values

Next to our chosen initialization strategy $w_{ij} \sim \mathcal{U}_{[0,1]}$ we test three different strategies and report the results (averaged over 50 tires and 95% confidence interval) in Figure 3. We test the following strategies:

- **Zero**: Initialize all weights to zero.
- **Gaussian**: We initialize all weights by sampling from two Gaussian distribution $\mathcal{N}(0;1)$ and $\mathcal{N}(0;0.01)$. In contrast to the chosen initialization with the uniform distribution strategy, this allows for negative weights and has no strict boundaries.
- **Gaussian with *Right* preference**: Using the knowledge of 2.3 we see that the action *Right* plays an important role. Hence, the weights are initialized with a Gaussian distribution, but setting the mean differently for the three actions: $\mu_0 = -1$, $\mu_1 = 0$ and $\mu_2 = 1$.

Initialization strategy	**Average total reward**	**95% confidence interval**
Uniform in $[0, 1)$	-147.9	± 7.1
Zero	-144.5	± 6.4
Gaussian with $\mu = 0$ and $\sigma = 1$	-152.5	± 7.4
Gaussian with $\mu = 0$ and $\sigma = 0.01$	-145.9	± 6.7
Gaussian with *Right* preference	-144.6	± 6.5

Table 3: Different initialization strategies for w

As shown in 3 the initialization strategy does not have a major impact on the average total reward. All values are close together and within each others confidence interval.

2.4.3 (3) Own exploration strategy

The exploration strategy that was utilized throughout the training is ϵ-greedy policy. A known drawback of this policy can be found when the exploration occurs. The policy picks indiscriminately from all possible actions. This means actions that lead to bad Q-function values are equally likely to get picked as good actions. Hence, we use the soft-max action selection rule, when exploring. Let P_a be the probability to pick action a in the case exploration occurs, then P_a is defined as

$$P_a = \frac{e^{Q^{(t)}(s,a)/\tau}}{\sum_b e^{Q^{(t)}(s,b)/\tau}},$$

where τ is the temperature parameter. For high temperatures, all actions are equally probable. If the temperature is low, then we pick the action greedily w.r.t the current estimates of Q. Therefore, the distribution of the probabilities of selecting a given action is changed from the uniform distribution.

The exploration policy was constant ($\epsilon = 0.001$) while varying the temperature values. The average reward is shown in Figure 10. Best performance was found for $\tau = 1$. However, since ϵ is small, the effect is not significant.

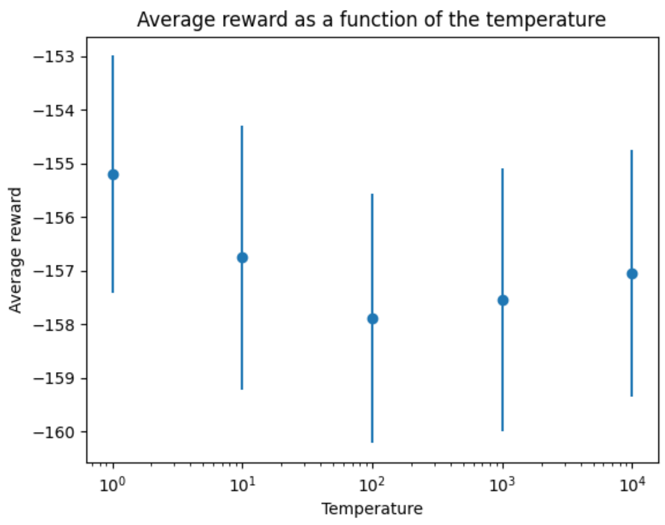
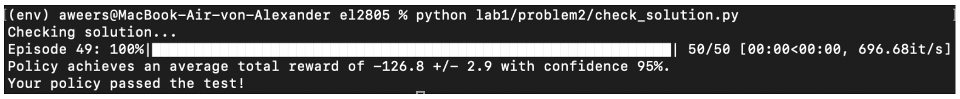


Figure 10: Average reward over 10000 episodes for soft-max exploration policy for different temperature values τ =[1 10 100 1000 10000].

2.5 3(f): Validated solution



```
(env) aweers@MacBook-Air-von-Alexander el2805 % python lab1/problem2/check_solution.py
Checking solution...
Episode 49: 100%|██████████████████████████████████████████████████████| 50/50 [00:00<00:00, 696.68it/s]
Policy achieves an average total reward of -126.8 +/- 2.9 with confidence 95%.
Your policy passed the test!
```

Figure 11: Output of `check_weights.py` with the saved parameters from our training. We clearly pass the required reward of at least -135.

As described, we saved the values of $\mathbf{w}$ and η after a successful training and provided them as an input to `check_solution.py`. As shown in 11 the output of the check is positive. The corresponding `weights.pkl` file is provided for reproducibility.