EL2805 Reinforcement Learning

Alice Karnsund
Therese Stålhandske

Abstract

This is the Lab Assignment 1 for the course Reinforcement Learning EL2805.

In this assignment we solve Markov Decision Processes for three different environments. We will handle both finite and infinite time horizons, with known or unknown probability transition matrix. The goal of each environment is to find the optimal policy for an agent, moving in a grid world.

An MDP is defined by set of $\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}$ where

- $\mathcal{S}$: State Space, all possible states s
- $\mathcal{A}$: Action Space, set of all possible actions a
- $\mathcal{R}$: Rewards for being in a state
- $\mathcal{P}$: Probability transition matrix

1 The Maze and the Random Minotaur

Consider a grid world of size 5×6 representing a maze with only one exit. An agent is trying to find his way out of the maze, at the same time as a Minotaur is roaming the around trying to eat him. What would be the optimal policy for the agent to follow at any given time step? To avoid being eaten, and at the same time try to exit the maze as fast as possible. We solve the problem with two different metrics in mind,

1. Maximize the probability of exiting in the maze before $T = 15$.
2. Minimizing the expected time of exiting the maze

1.1 Environment description

Here we describe the environment and the MDP specifications for each one of the intended metrics. Both these scenarios have similar MDPs, with a few alteration, more closely described below.

1.1.1 Maximizing the probability of exiting the maze

The state space constitutes of all possible combination of our agent and the Minotaur's moves. As both the agent and the Minotaur can be in 30 possible states each, the size of the state space is $30 \times 30 = 900$.

At any given situation the agent can choose to perform an action in the Action space $LEFT, RIGHT, UP, DOWN, STAY$. This is always possible, except when then grid is blocked by either a wall or the edge of the maze. The Minotaur moves are decided uniform randomly over the Action space. For the Minotaur, two version of the action space will be explored, $\mathcal{A} = LEFT, RIGHT, UP, DOWN, STAY$ and $\mathcal{A} = LEFT, RIGHT, UP, DOWN$ (one case when the Minotaur is allowed to stay at a grid for more than one time step, and one when he is not). The Minotaur can go through the walls, but he is restricted within the edges of the maze.

The transitions are described in matrix format, 30×30 for each possible choice of action for the agent. For each pair of the agent's location in grid and his chosen action, the next state can be any of 2-4 other possible states, depending on the Minotaur's next possible actions. The probability of transition to any of these states, is given by $\frac{1}{\textit{possible moves of Minotaur}}$.

If the agent is at the same position as the Minotaur, the game is over and a terminal state has been reached. The same applies if the agent reached the end of the maze, the game is then over and a terminal state is reached. As we wish to avoid the Minotaur, being eaten will result in a negative reward of -10. Further, to maximize the probability of getting out of the maze, a negative reward is given if the agent goes to a state which might be in the Minotaur's path. When the agent is in a state not next to/on the Minotaur or the terminal state, the reward is 0. Reaching the end of the maze, gives a reward of $+10$.

We are working with a finite-horizon $T < \infty$ and the objective is to maximize the probability of exiting the maze. To solve this problem, we will make use the Bellman's equations for the value function given by,

$$V_t(s_t) = max_{a \in A_{s_t}} [r_t(s_t, a) + \sum_j p_t(j|s_t, a) V_{t+1}(s_t, a, j)] = max_{a \in A_{s_t}} Q_t(s_t, a) \qquad (1)$$

An optimal policy π is obtained by selection $\pi_t(s_t)$ at time t such that

$$Q_t(s_t, \pi_t(s_t)) = max_{a \in A_{s_t}} Q_t(s_t, a) \qquad (2)$$

1.2 Results

Bellman's equations are then solved with dynamic programming, starting in the terminal state and iterating backwards to the initial state.

The optimal policy for some fixed Minotaur states are demonstrated below, in figure 1 - 2. Each one of the arrows, represent an optimal action in that state and · represent action STAY. Notice that the arrows in the pictures, doesn't necessarily demonstrate the only optimal policy in each state, but for simplification only one action is shown. In these examples, the Minotaur is not able to stay in the same position. If the agent is in an adjacent grid to the Minotaur, the best possible action might be taking the Minotaur's previous spot, if it moves us closer to the exit. When the Minotaur is allowed to choose the action "STAY", the best policy is not necessarily going to the Minotaur's current state. In this case, probability of exiting the maze reduces, as the Minotaur's movements gets more unpredictable and the possibility of the Minotaur "blocking" a path gets larger. With a smaller T, it then gets harder for the agent to exit the maze in time.

↓	·	→	↓	←	←
·	↓	→	↓	↑	↑
→	M	→	↓	→	↓
·	→	→	→	→	↓
↑	←	←	←	B	←

Figure 1: Optimal policy for the agent, when the Minotaur is in state [2,1], as demonstrated with M in the grid.

↓	↓	→	↓	←	←
↓	↓	→	↓	↑	↑
↓	↓	→	↓	·	↓
↓	→	→	→	→	M
↑	←	←	←	B	←

Figure 2: Optimal policy for the agent, when the Minotaur is in state [3,5], as demonstrated with M in the grid.

Figure 3 below shows the agent's actions during a simulation with $T = 15$. When a · appears before an arrow, it means that the agent choose to stay for one time step and then starts moving in the arrow direction. The agent tries to avoid the Minotaur, causing it to take a detour or to stay in the same location for more than one time step. With this policy, the average probability over 10 000 simulations of exiting the maze when $T = 15$ and the Minotaur is not allowed to choose the action "STAY", is $p(\textit{exiting}|\textit{not stay}, T = 15,) \approx 0.98$. The corresponding average probability over 10 000 simulations, when the Minotaur is allowed to choose the action "STAY" is $p(\textit{exiting}|T = 15, \textit{with stay}) \approx 0.75$. If the agent does not exit the maze, the reason will not be because the Minotaur ate him, but due to the fact that time runs out. This can happen for example when the Minotaur is continuing to block the way for more than one time step.

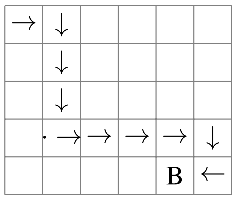

Figure 3: Simulation of the agents move when the Minotaur is not allowed to stay.

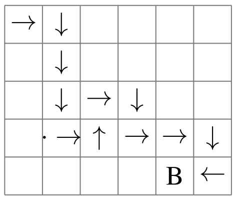

Figure 4: Stimulation of the agents move when the Minotaur is allowed to stay.

The probability of exiting the maze as a function of the maximum time allowed (T) is shown in Figure 5 below, here the Minotaur is not allowed to choose the action STAY. When $T > 15$, the probability of getting out the maze gets close to 1, meaning that the agent will always find its way out of the maze. This is shown in contrast to the probability of exiting the maze when the Minotaur is allowed to choose action STAY, in figure 6.

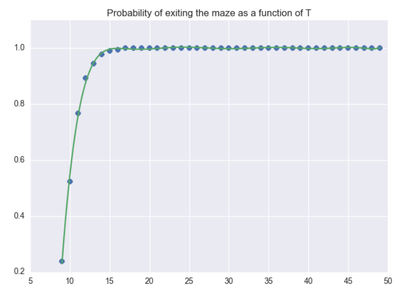

Figure 5: Probability of exiting the maze as a function of the maximum time T allowed in the maze, the Minotaur is not allowed to STAY. When T gets lager, the probability of exiting the maze increases and gets close to 1 when $T \approx 15$ and above.

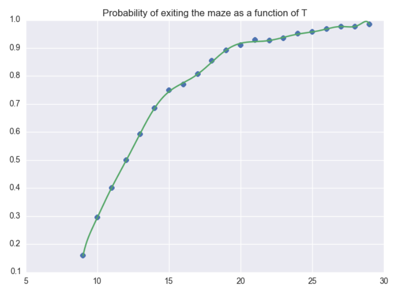

Figure 6: Probability of exiting the maze as a function of the maximum time T allowed in the maze and the Minotaur is allowed to STAY. This makes the environment harder to predict, resulting in a lower probability of exiting the maze

1.2.1 Minimizing expected time in the maze

The MDP of this problem is similar to the one described in the section above. We explore the same state space with the same action space. Also, the transition matrix will not be altered.

Now the goal is to minimize the expected time of exiting the maze. At each time step when the agent is not in the terminal state the reward is thus set to -1. If the agent tries to go outside the maze or go into a wall, the state remains the same as the previous step with a reward of -1. In this way the agent will learn how to navigate the maze without collecting unnecessary negative rewards. When the agent has reached the terminal state the reward given is 0. This to motivate the agent to finish as fast as possible. Also, this agent is not as cautious as the one in the previous example. The average probability over 10000 runs of exiting the maze with this policy is then $p \approx 0.66$ with the average time of $12.9 \approx 13$ time steps with the Minotaur allowed to stay still. With the Minotaur not allowed to stay still, the probability of exiting the maze is $p \approx 0.74$ and the expected time before exiting the maze is $9.9 \approx 10$ time steps.

2 Bank robbing

Consider a grid world of size 3×6, representing a town in which a robber is trying to rob the banks with a police chasing after him. At any time the robber is aware of the police's position and decides on an action, based on this and the locations of the banks. We will solve this problem maximizing the expected rewards.

2.1 Environment

The number of states are described as a pair, the police and the robber positions. The total number of states then becomes, $18 \times 18 = 324$.

The robber can at any state choose between the actions STAY, UP, RIGHT, LEFT and DOWN expect when there is an edge blocking the way. The police chooses a random action, in a way that the police always will move towards the robber choosing from UP, RIGHT, LEFT and DOWN.

The transition probabilities can be described as follows: If the robber and the police are at the same position (the police catches the robber). No matter the action, the probability of getting in the initial state is 1, $P(initialstate|agent_grid = police_grid, x)$ for $x =$ STAY, UP, RIGHT, DOWN or LEFT. For each pair of the robber location in grid and his chosen action, the next state can be any of the 2-4 other possible states, depending on the police's possible actions. The probability of transition to any of these states is given by $\frac{1}{\#\textit{of possible moves of police}}$

If the robber is on the bank, he receives a a reward of $+10$ during each time-step he is there. If the polices catches the robber, he gets a reward of -50 and the game is reinitialized, with the robber and the police are reverted to their initial state. The game then continues. This problem has an infinite time-horizon with a discount λ.

The MDP is solved with Value Iteration to find the optimal policy for the agent.

2.2 Results

When λ is close to 1, the value function consider a longer time horizon. As shown in the example below in Figure 7, this means that the robber have a stronger tendency of moving to the other side of the grid world. This because the robber then will be further away from the police, so it will have some extra time of robbing the banks on the right side. Wheres in the case when lambda is small, the agent has a more limited time horizon and will behave more short minded, always staying on the same side of the grid world.

$\downarrow$	$\cdot$	$\leftarrow$	$\rightarrow$	$\rightarrow$	$\cdot$
P	$\leftarrow$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\downarrow$
$\uparrow$	$\cdot$	$\leftarrow$	$\rightarrow$	$\rightarrow$	$\cdot$

$\lambda = 0.01$

$\downarrow$	$\cdot$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\cdot$
P	$\leftarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\downarrow$
$\rightarrow$	$\cdot$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\cdot$

$\lambda = 0.99$

Figure 7: Optimal policy for the agent in each state with a fixed police position at [1,0]. The left grid shows the optimal policy for a small λ and the right grid a policy when λ is close to 1.

In figure 8, we see the value function of the initial state of the agent and the police as a function of the discount rate λ. When λ gets closer to one, the value function grows to infinity. This is because we have an infinite MDP, where the expected reward will go to infinity as we don't have a terminal state.

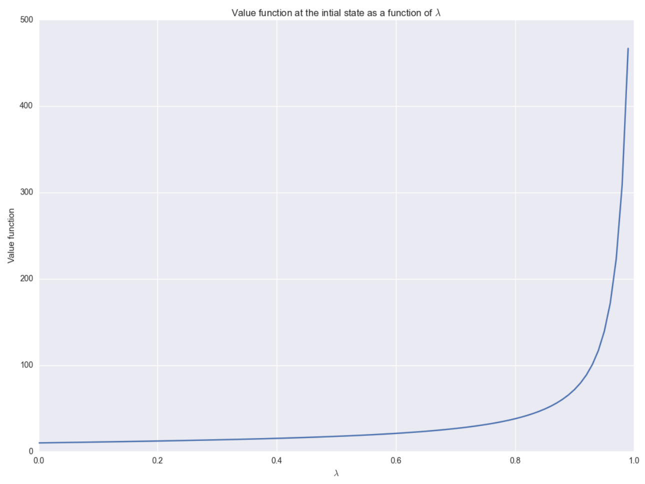

Figure 8: Value function of the initial state $[0, 0]$, $[1, 2]$ as a function of the discount rate λ

3 Bank Robbing (Reloaded)

Considering a grid world of size 4×4, which in this case represents a new town. In this world a robber trying to heist a bank as many times as possible without getting caught by the police. Our goal is to develop two algorithms, Q-learning and SARSA, that learns a policy that maximizes the total discounted reward for a discount factor $\lambda = 0.8$.

3.1 Environment

R			
	B		
			P

Figure 9: Start grid

Figure 1 shows the initial state in this problem, R is the position of the robber, P the position of the police and B the position of the bank. In each time step, the police chooses randomly among his actions to perform, up, down, left and right. The robber can choose between the same actions as the police, but can also choose to stand still.

$$ACTIONS_{Robber} = [STAY, UP, DOWN, LEFT, RIGHT]$$
$$ACTIONS_{Police} = [UP, DOWN, LEFT, RIGHT]$$

If the robber get caught by the police, he receives a negative reward of -10, if he robs the bank he receives a positive reward of 1, and otherwise he will get a reward of 0.

3.2 Method

Both the algorithms we are considering here makes use of Temporal-Difference (TD) Learning methods. TD learning is a combination of Monte Carlo and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based on other learned estimates, without waiting for a final outcome (they bootstrap) [1]. Both algorithms tries to estimate the optimal

[1]Richard S. Sutton and Andrew G. Barto, Reinforcement Learning An Introduction, 2nd edition, (Cambridge, MA: The MIT Press, 2018), p.119

action-value function Q, which is the maximum expected reward starting from state s and taking action a,

$$Q(s,a) = r(s,a) + \lambda \sum_j p(j|s,a) max_b Q(j,b) \tag{3}$$

In comparison to problem 1, we are dealing with an infinite time horizon and also, we do not know the model. Therefore we make use of the Q-learning and SARSA algorithms to approximate the desired action-value function Q.

3.3 Implementation

We used `Python3` to implement both algorithms.

Q-learning

Q-learning learns the true Q-function, and hence the optimal policy provided that the behavior policy explores all (state, action) pairs infinitely often. Q-learning approximates the true Q-function in the following way. At time t, we have an estimated Q-function, $Q^{(t)}$. The system is in state s_t and action a_t is selected (under the behaviour policy π_b). We observe s_t, a_t, r_t, s_{t+1} and update Q accordingly,

$$Q^{(t+1)} = Q^{(t)}(s,a) + 1_{(s_t,a_t)=(s,a)} \alpha_{n^{(t)}(s_t,a_t)} [r_t + \lambda max_{b \in A} Q^{(t)}(s_{t+1}, b) - Q^{(t)}(s_t, a_t)] \tag{4}$$

For this algorithm the robber explores actions uniformly at random.

SARSA

SARSA on the other hand, learns the Q-function of the current behaviour policy. It learns $Q^{\overline{\pi}}$, the (state, action) value function including the exploration phase (e.g. ϵ-greedy)[2]. At time t, we have an estimated Q-function, $Q^{(t)}$. The system is in state s_t and an action a_t is selected (under π_t ϵ-greedy w.r.t $Q^{(t)}$). We observe s_t, a_t, r_t, s_{t+1} and from s_{t+1} we also observe a_{t+1}. We then update Q accordingly,

$$Q^{(t+1)} = Q^{(t)}(s,a) + 1_{(s_t,a_t)=(s,a)} \alpha_{n^{(t)}(s_t,a_t)} [r_t + \lambda Q^{(t)}(s_{t+1}, a_{t+1}) - Q^{(t)}(s_t, a_t)] \tag{5}$$

Here we make use of an ϵ-greedy exploration. Meaning that the algorithm selects actions uniformly at random with a probability ϵ, and chooses the action that maximizes the Q value with probability $1 - \epsilon$,

$$a = \left\{ \begin{array}{cc} \text{argmax}_b Q^{(t)}(s,b), & \text{w.p } 1-\epsilon \\ \text{uniform}(A_s), & \text{w.p } \epsilon \end{array} \right\} \tag{6}$$

In both cases, $n^{(t)}(s,a) \equiv \sum_{m=1}^{t} 1[(s,a) = (s_m, a_m)]$. That is, $n(s,a)$ is the number of updates of $Q(s,a)$.

3.4 Results

Both algorithms used a discount factor $\lambda = 0.8$ and were run over 10.000.000 iterations.

Q-learning

In Figure 10, two plots for the value function in different states over time are shown. To the left is a plot for the initial state and to the right for the 149th state, which is when the robber is at the bank and the police is just below (index $[2,1]$). In both cases we see a clear convergence of the value function.

[2]Alexandre Proutiere, EL 2805 – Reinforcement Learning, Part 4: Q-learning and SARSA algorithms, (KTH, The Royal Institute of Technology), p.43

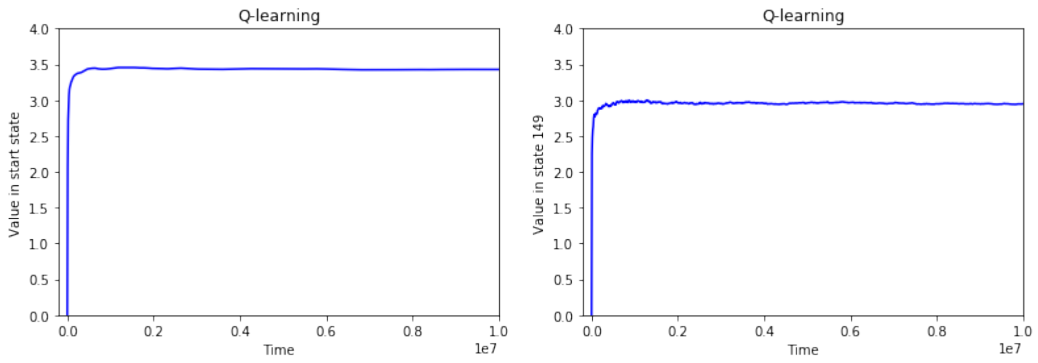

Figure 10: Plot of the value function over time for the initial state (left) and state 149 (right).

In Figure 11, two different policies for the robber are shown. The left one shows the situation when the police is at the bank and the right one shows the situation when the police is at position $[2, 3]$. From the left policy we note that the robber has learned that, when standing in the same row or column as the police, the optimal strategy is to "switch place" with the police. And if the robber stands one diagonal step away, the best strategy is to stand still, to make sure that there is no chance for the police to catch him in the next time step. From the same policy, we can see that if the robber is in the same row or column as the police, but two steps away, it is again a good idea to stand still and wait and see what move the police will do. In the policy to the right, we see that the robber will stay at the bank in every time step as long as the police is not disturbing. In both cases we see that the bank works as a "sink".

·	↓	·	↓
→	PB	←	·
·	↑	·	←
↑	·	←	←

→	↓	↓	←
→	B·	←	←
↑	↑	↑	P
↑	↑	←	←

Figure 11: Policies for when the police is at the bank (left) and at position [2,3] (right).

Note, there might be more than one optimal action in each state. The above figure just shows one optimal action in each state.

SARSA

To make sure that $Q^{(t)}$ tends to the true Q-function as $t \to \infty$, we must decrease epsilon over time, i.e $\epsilon \to 0^+$ as $t \to \infty$. In this case we used a linear decrease,

$$\epsilon_t = \epsilon + \frac{\epsilon/C - \epsilon}{N_{iteration}} t \tag{7}$$

Here C is a constant, which we set to 100 and 1000. $N_{iteration}$ is the number of iterations used, in this case $10.000.000$. For ϵ, we chose $0.1, 0.3$ and 0.5.

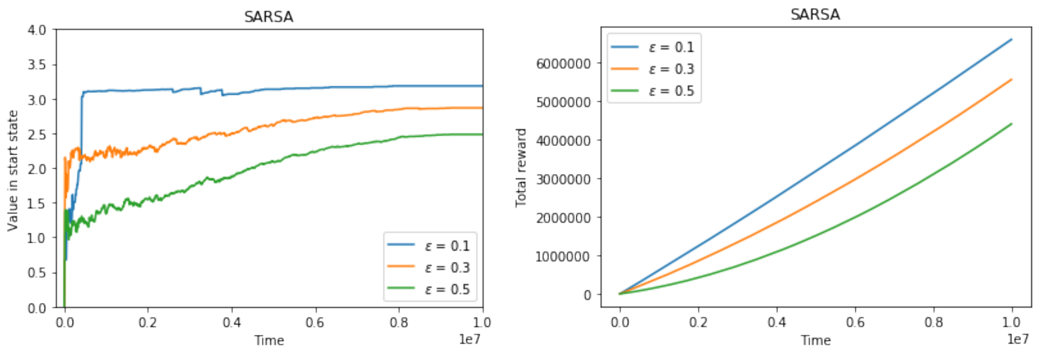

Figure 12: Plot of the value function over time for the initial state (left) and the corresponding cumulative rewards (right). Here $C = 100$ in equation 7.

In Figure 12, to the left, we see a plot of the value function in the initial state for three different ϵ. We see that the lower the value of ϵ, the higher the value of the value-function. After $10.000.000$ iterations all three value functions have converged, as can be seen in the figure.

In Figure 12, to the right, we see a plot of the corresponding cumulative rewards. In all three cases the cumulative reward steadily increases. If we use a small ϵ we see that we have a higher reward all through. This is expected for this case, since a smaller ϵ will take less random actions and will thus focus more on taking actions that will yield a good reward. This of course depends a lot on how large the state space is and the variations in rewards.

→	↓	←	←
→	B·	←	←
↑	↑	↑	P
→	↑	·	←

→	↓	←	←
→	B·	←	←
↑	↑	↑	P
↑	↑	←	←

↓	↓	←	←
→	B·	←	←
↑	↑	←	P
↑	↑	←	←

Figure 13: Policies for when the police is at the bank, for $\epsilon = 0.1$, $\epsilon = 0.3$ and $\epsilon = 0.5$ respectively.

In Figure 13 we see three policies, one for each ϵ, for the state when the police is at position $[2, 3]$. As can be seen, these policies differs a little bit. This is probably due to the fact that, in some positions, more than one action is given the same value or, depending on ϵ, some actions might have slightly higher value in certain states. The policy for $\epsilon = 0.3$ has only one action different from the Q-learning case.

4 Conclusion

For the first two problems, we were given a model and therefore we could make use of Bellman's equations and Value iteration to find the optimal policy. In the last problem, no model where given, and thus we had to learn the model by approximating the Q-function in two different ways. Comparing these two cases we can draw the conclusion that, even though the model is not know, we are able to find optimal policies by the use of Q-learning and SARSA.