Defining a Markov Decision Process Adherenet Environment

Ziad Al-Ziadi and **Behzad Javaheri**

1. Introduction

Reinforcement learning (RL) is a machine learning method allowing a learner (agent) to dynamically interact with its environment and take a sequence of actions. The environment, in return, provides rewards (+/-) and a new state based on previous action. The agent is not directed to take any particular action, rather to seek actions to maximise rewards. The process starts from random trials aiming to complete with refined actions leading to higher rewards, achieved primarily through exploration of the environment and subsequent exploitation [1].

RL differs from supervised/unsupervised learning due to the dynamic nature of the interaction. In supervised learning, an agent is directed to take actions to maximise reward using a labelled training dataset. In RL, however, an agent is directed to utilise a reward mechanism, without having prior access to data, to select an action. The difference with unsupervised learning arises as unsupervised learning seeks to find hidden structure within unlabelled data [2].

The common framework for solving RL problems and sequential decision making is Markov Decision Processes (MDP). This is a fundamental formalism and intuitive mathematical framework to describe RL environment [1-3], modelled as a set of states and actions to control the systems state to maximise performance measure, rewards. Problems including planning, learning robot control and game playing have successfully been modelled as MDP [4].

Briefly, the agent and the environment interact at each time step, [t 0, 1, 2, 3…] and the agent obtains information about environment state St. Based on St at time t, the agent selects action A_t that is available in St. The agent responds by randomly moving to a new state (S') and consequently receiving a numerical (+/-) reward [2]. Both R_t and S_t with discrete probability distributions are dependent on previous action and state in line with MDP. This suggests that transition into S' is dependent on action A and the current state S [2].

For task 1, we will design, implement and describe an MDP adherent tabular environment.

2. Description of the environment

We have defined a 5x5 discrete-space grid environment with 25 possible states identifiable by row and column index. There are 4 pre-defined states. Our agent, cheese, trap and end-goal stochastically initialised at the beginning of each episode. For example, at a particular state represented in Figure 1, our agent is in C_0 (0, 0), cheese in C_{12} (2, 2), trap C_{18} (3, 3) and end-goal in C_{24} (4, 4).

C_0	C_1	C_2	C_3	C_4
C_5	C_6	C_7	C_8	C_9
C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
C_{15}	C_{16}	C_{17}	C_{18}	C_{19}
C_{20}	C_{21}	C_{22}	C_{23}	C_{24}

Figure 1. A grid world (5x5) to solve an RL problem.

3. Description of the agent and its action

Our environment encompasses a single agent; for simplicity is represented as a mouse (Fig. 1). The agent is able to take 4 possible actions and move one cell at a time: up, right, down or left. The agent cannot move diagonally; only moves in the horizontal and vertical axis.

Given that the agent moves from one state to another, it must make decisions based on the available actions at its current state. Furthermore, the other states on the grid are defined by their actions and the next state. The grid exhibits a finite array of states which can be defined as $S = s_1, s_2, s_3, \ldots, s_N$. and within each state, a finite array of set actions A_t existing where $A_t = [a_1, a_2, \ldots, a_N]$. Both S and A_t are utilised as a part of the state transition matrix given by: $P^a_{ss'} = \mathrm{P}[S_{t+1} = S' \mid S_t = s, A_t = a]$ indicating the probability of transition from state S to state S' given some action a (Table 1).

State(s)	Action(a)	Next state (S_{t+1})
C_0	*Left*	C_1
C_1	*Up*	C_1
C_2	*Down*	C_7

Table 1. Transition of 3 states with

4. Description of different dynamics of the environment and rewards

The rules governing this grid environment is simple. The agent starts at state C_0, interacts with the environment and at each step has 4 possible actions, as described. If there is a wall in the direction of the agent, (e.g., moving "up" in state C_0) the agent remains in the same state. The goal of the agent is to finally reach the end goal at C_{24} whilst avoiding trap terminal state (C_{18}).

Given a transition to some state S, the environment will return an immediate reward in which the agent aims to maximise the accumulated rewards over time. The reward function is represented by *R* and can be defined as*:* $R_s = \mathrm{E}[R_{t+1} \mid S_t = s]$.

For each step, apart from cheese, trap and end-goal, a small negative reward of -1 is awarded. This negative reward is to prevent the agent to loop in the environment to maximise its awards without ever reaching the goal. This encourages the agent to find the shortest possible route (to avoid high negative rewards). Big rewards are assigned to terminal states of trap (-100) and end-goal (+100). The intermediate reward cheese state carries an award of +4.

5. Learning Policy

Formally, the learning policy defines how the agent behaves in the environment and can be described as a mapping of the probability of taking action *A* whilst at state *S*. When interacting with the environment for the first time, the agent must follow a random policy so that it can learn whether the actions it takes are positive or negative [2].

In this task, we implement both a stochastic as well as an epsilon *e-greedy* policy in which the agent will choose an action with a *1 - e* probability to encourage exploration to learn its available actions. In stochastic policy, the mapping does not directly point a state to action but instead the state to a probability distribution over an action space [2]. The agent will perform different actions for a given state based on the probability distribution returned by the stochastic policy. The actions that the agent selected can be represented by the mapping π: $A \times S \rightarrow [0, 1]$ where $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$ where π is the policy.

References

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237-285, 1996.

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

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific, 1996.

[4] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement learning*: Springer, 2012, pp. 3-42.