CS 4080-5080 - Reinforcement Learning
Homework 1 Report

Carlos E. L. P. X. Torres
University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, Colorado 80918
clopespi@uccs.edu

Abstract

Homework 1 Report for the class CS 4080-5080 - Reinforcement Learning due to 09/29/2021. On this assignment, it is implemented a reinforcement learning algorithm to teach an agent how to navigate in a 5x5 maze from a start state to the goal state, using a Monte Carlo method to start learning from a random approach and slowly improving the optimal policy using a greedy approach and stop when the policy does not improve anymore. A simulation is presented showing the implemented algorithm in action and also showing a chart of the learning improvement progress.

1 Introduction

Homework 1 Report for the class CS 4080-5080 - Reinforcement Learning. On this assignment, it is implemented a reinforcement learning algorithm to teach an agent how to navigate in a 5x5 maze from a start state to the goal state. No reinforcement learning libraries are used, only some common libraries, like for numeric processing (numpy), math, random, and time.

On section 2, it is presented a model of the problem, describing the various components required for the agent and the environment (states, rewards, actions, barriers, edges avoiding handling).

On section 3, a first random policy is presented, where every action that is possible in a state is equally probable.

On section 4, it is presented the algorithm for learning the optimal policy with a Monte Carlo method with the equi-random method. Then improve the random policy slowly using a greedy approach and stopping when the policy does not improve anymore.

On section 5, it is presented how the algorithm was implemented and a simulation was created to illustrate it.

In conclusion, it is discussed the final considerations, problems faced, and how they were overcame during the work.

2 Problem Model

To model the 5x5 maze (Figure 1) as a reinforcement learning problem, we should formulate it in terms of a Markov Decision Process (MDP), consisting of a tuple $\{S, A, P, R, \gamma\}$ (Sutton and Barto 2018), where:

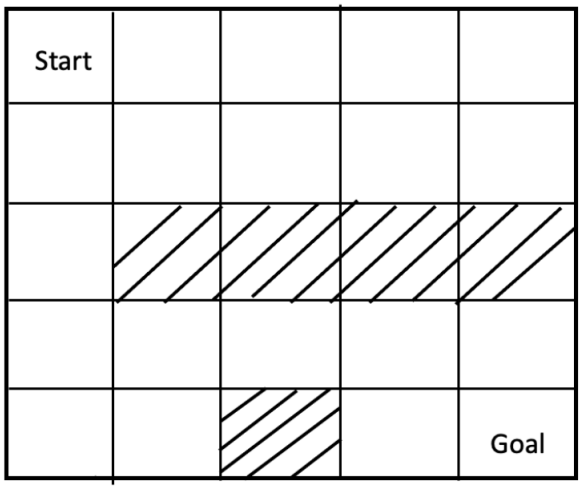

Figure 1: The 5x5 Maze

- *S* - The state space: a set of 25 states represented by a 5x5 2D matrix of cells (x,y) identifying each possible state inside the maze. The start state is represented as (0,0) and the goal state (4,4).
- *A* - The action space: a set of 4 possible actions $\{N, S, E, W\}$ that the agent can take in each state inside the maze, each one being a directions (North, South, East, West). In this problem, not all actions are allowed in all states, due to the barrier states and the boundaries of the maze.
- *P* - The transaction probability function: defines how the agent can change states inside the maze. The function will serve to define the allowed actions inside each state (or cell) of the maze, associating probabilities to each state-action pair, and 0 (zero) for the ones that are not allowed. This will also serve to define the barriers inside the maze, by assigning 0 to an action that will lead to a barrier state. See Table 1 for the representation of this function.

The barrier states themselves (s_{12}, s_{13}, s_{14}, s_{15}, s_{23}) will have a probability of 0 (zero), to keep the integrity of the function. Similarly, the goal state itself (s_{25}) will

State / Action	N	S	E	W
s_1 (0,0)	0	0.5	0.5	0
s_2 (0,1)	0	0.33	0.33	0.33
s_3 (0,2)	0	0.33	0.33	0.33
s_4 (0,3)	0	0.33	0.33	0.33
s_5 (0,4)	0	0.5	0	0.5
s_6 (1,0)	0.33	0.33	0.33	0
s_7 (1,1)	0.33	0	0.33	0.33
s_8 (1,2)	0.33	0	0.33	0.33
s_9 (1,3)	0.33	0	0.33	0.33
s_{10} (1,4)	0.5	0	0	0.5
s_{11} (2,0)	0.5	0.5	0	0
s_{12} (2,1)	0	0	0	0
s_{13} (2,2)	0	0	0	0
s_{14} (2,3)	0	0	0	0
s_{15} (2,4)	0	0	0	0
s_{16} (3,0)	0.33	0.33	0.33	0
s_{17} (3,1)	0	0.33	0.33	0.33
s_{18} (3,2)	0	0	0.5	0.5
s_{19} (3,3)	0	0.33	0.33	0.33
s_{20} (3,4)	0	0.5	0	0.5
s_{21} (4,0)	0.5	0	0.5	0
s_{22} (4,1)	0.5	0	0.5	0
s_{23} (4,2)	0	0	0	0
s_{24} (4,3)	0.5	0	0.5	0
s_{25} (4,4)	0	0	0	0

Table 1: Transition probability function representation

also be considered with probability 0 (zero). Because once achieved, the agent should not go anywhere else, and the episode ends.

- *R* - The reward signal: represented by a set of rewards from the environment to the agent. For this problem, two types of rewards will be used. For every step in the maze, the agent receives a reward of $-0.1/(\text{number of cells})$, where number of cells (states) is 25. And when the agent reaches the goal state, a reward of 1 is given. The negative rewards will make the agent prefer a policy with fewer steps to achieve the goal state.
- γ - The discount factor: can assume values $0 \leq \gamma \leq 1$. It increases the interest of the agent in having earlier rewards instead of future or later rewards. For this problem, it will be set initially to $\gamma = 0.99$, and can change during the learning process to see how it affects the algorithm.

3 Initial Random Policy

From the transition probability function (Table 1), we can define the initial equi-random policy for each state. For example, for state s_1 it will look like:

$$\pi_0(\text{N}|s_1) = 0,\ \pi_0(\text{S}|s_1) = 0.5,\ \pi_0(\text{E}|s_1) = 0.5,\ \pi_0(\text{W}|s_1) = 0$$

The same can be applied to all states to also get their initial policy. They all together will make the initial equi-random policy for our algorithm.

4 Algorithm

Here is presented the algorithm for learning the optimal policy with a Monte Carlo method. The method used is a Monte Carlo control, without exploring starts (because we have a fixed starting state), on-policy with ϵ-greedy policies (ϵ-soft). Given that the algorithm starts with an initial equi-random policy (greater than zero) and slowly improves it using a greedy approach and stops when the policy does not improve anymore.

Algorithm 1: Monte Carlo on-policy control with ϵ-soft

1: **Parameters**:
   $\epsilon > 0$
   $0 \leq \gamma \leq 1$
   numberOfEpisodes > 0
2: **Initialize**, for all $s \in S, a \in A(s)$:
   $Q(s,a) \leftarrow$ arbitrary
   $Returns(s,a) \leftarrow$ empty list
   $\pi \leftarrow$ an arbitrary ϵ-soft policy

3: **for** $i \leq$ numberOfEpisodes **do**
4: Generate an episode using π
5: $G \leftarrow 0$
6: **for** $(S_t, A_t, R_t) \in$ episode **do**
7: $G \leftarrow \gamma G + R_t$
8: **if** $(S_t, A_t) \in$ episode **then**
9: Append G to $Returns(S_t, A_t)$
10: $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
11: $A^* \leftarrow argmax_a Q(S_t, a)$
12: **for all** $a \in A(S_t)$ **do**
13: **if** $a = A^*$ **then**
14: $\pi(a|S_t) \leftarrow 1 - \epsilon + \epsilon/|A(S_t)|$
15: **else**
16: $\pi(a|S_t) \leftarrow \epsilon/|A(S_t)|$
17: **end if**
18: **end for**
19: **end if**
20: **end for**
21: **end for**

22: **Return** π

The algorithm receives three parameters (Line 1): ϵ, γ, and numberOfEpisodes. The initial ϵ value that is being used is 0.01. The γ is set to 0.99. And the numberOfEpisodes is set to 100.

It starts by initializing (Line 2) $Q(s,a)$ to an arbitrary (zero filled) structure to store the rewards for the (s,a) pairs. The $Returns(s,a)$ as an empty list. And the initial equi-random policy π.

On Line 3, starts the main loop of the algorithm, using the numberOfEpisodes parameter. In each iteration, it is generated an episode based on the policy π (Line 4). On Line 6, starts another loop for each step of the episode, that will have (S_t, A_t, R_t). Inside it, the total reward G is incremented with R_t and discounted by γ (Line 7). Then, if (S_t, A_t) is in the episode, G is appended to $Returns(S_t, A_t)$ (Line 9), $Q(S_t, A_t)$ is updated with the average of $Returns(S_t, A_t)$

(Line 10). Then it is obtained the action A^* with the maximum value on $Q(S_t, a)$ (Line 11).

On Line 12, a loop for the actions a inside $A(S_t)$ is created. Inside it, if $a \neq A^*$ (all nongreedy actions), then the policy $\pi(a|S_t)$ receives the minimal probability of selection $\epsilon/|A(S_t)|$. Otherwise, if $a = A^*$ (the greedy action) it receives the remaining bulk of probability $1 - \epsilon + \epsilon/|A(S_t)|$.

5 Implementation and Simulation

The algorithm was implemented in Python 3 using only native or common libraries like: random, time, sys, a numeric processing library called numpy, and a simulated environment for the simulation, using a custom Open AI gym (Brockman et al. 2016) to model the maze, called gym_maze (Chan 2020).

The algorithm starts with the initial equi-random policy, described on section 3, and slowly improves the policy after each iteration (episode). Each time, the policy changes a little with slightly better values for the state-action pair.

5.1 Policies Learned

After running the algorithm several times, using numberOfEpisodes = 100, a few different policies were learned. The policy that is generated is a dictionary with 25 states and, for each of them, the probabilities to take each of the 4 actions $\{N, S, E, W\}$ (in the dictionary, they are represented as 0,1,2,3 for simplicity). Then, to create the actual final policy (sequence of states and actions taken), the following calculation is taken to obtain the action with higher probability value:

$$\pi(s) = argmax_a Q(s, a) \tag{1}$$

Below, a few policies are shown:

Policy 1 (best one):
$\pi(s_1) = S \rightarrow \pi(s_6) = S \rightarrow \pi(s_{11}) = S \rightarrow \pi(s_{16}) = E \rightarrow \pi(s_{17}) = E \rightarrow \pi(s_{18}) = E \rightarrow \pi(s_{19}) = E \rightarrow \pi(s_{20}) = S \rightarrow \pi(s_{25})$

Policy 2 (second best):
$\pi(s_1) = S \rightarrow \pi(s_6) = S \rightarrow \pi(s_{11}) = S \rightarrow \pi(s_{16}) = E \rightarrow \pi(s_{17}) = E \rightarrow \pi(s_{18}) = E \rightarrow \pi(s_{19}) = S \rightarrow \pi(s_{24}) = E \rightarrow \pi(s_{25})$

Policy 3:
$\pi(s_1) = E \rightarrow \pi(s_2) = S \rightarrow \pi(s_7) = W \rightarrow \pi(s_6) = S \rightarrow \pi(s_{11}) = S \rightarrow \pi(s_{16}) = E \rightarrow \pi(s_{17}) = E \rightarrow \pi(s_{18}) = E \rightarrow \pi(s_{19}) = E \rightarrow \pi(s_{20}) = S \rightarrow \pi(s_{25})$

Policy 4:
$\pi(s_1) = E \rightarrow \pi(s_2) = E \rightarrow \pi(s_3) = S \rightarrow \pi(s_8) = W \rightarrow \pi(s_7) = W \rightarrow \pi(s_6) = S \rightarrow \pi(s_{11}) = S \rightarrow \pi(s_{16}) = E \rightarrow \pi(s_{17}) = E \rightarrow \pi(s_{18}) = E \rightarrow \pi(s_{19}) = S \rightarrow \pi(s_{24}) = E \rightarrow \pi(s_{25})$

The difference between the learned policies is that some of them take a few more steps to get to the goal. But these steps are not so disruptive, they are close or around the steps for the best policy. So, the algorithm is working as intended.

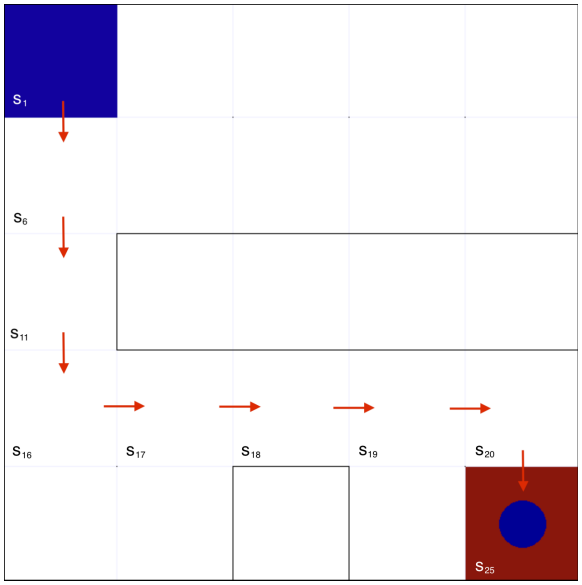

Figure 2: Policy 1 represented in the maze

Figure 3: Chart of cumulative rewards (G) during episodes

That can be also be checked when it is executed with more episodes, like 1000, which makes virtually always the best policy, with fewer and direct steps.

The algorithm starts converging after around 20 episodes of training, as shown on Figure 3. The chart shows the cumulative rewards (G) values changing during the episodes.

6 Conclusion

The problem of the 5x5 Maze proposed on this work is a great example to use reinforcement learning to solve. It can be clearly formulated as an MDP and the initial policy created using the transition probability function. The algorithm selected to solve it was Monte Carlo on-policy control with ϵ-soft, which showed a great performance and result.

The implementation of the algorithm was generally straightforward, with a few problems faced but overcame

after some thought. It was necessary to create some auxiliary functions to create the initial policy, prepare the states, create the state-action dictionary (Q), run the episodes in a manner that we could pass the current policy, store the cumulative rewards (G) during the iterations, and test the final policy generated at the end.

The algorithm main relevant parameters that were changed during the training were the number of episodes to run and the ϵ (epsilon) value. The number of episodes were set initially to smaller values, then increased gradually to understand how the policy was being improved and the cumulative rewards increased. A minimum of runs of 100 was selected to better visualize the results, reflected on Figure 3. The ϵ was initially set to 0.01 and increased along several runs. The higher this value is the longer the algorithm takes to analyze and improve the policy, because this number increases the probability of exploring more random options. For the state space of this problem, the initial value of 0.01 proved to be the best value to select.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *CoRR*, abs/1606.01540.

Chan, M. 2020. gym-maze, A basic 2D maze environment. https://github.com/MattChanTK/gym-maze. Accessed: 2021-09-17.

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