Programming Assignment 2

CS 747: Foundations of Intelligent & Learning Agents

(Autumn 2023)

Atishay Jain (210050026)

210050026@iitb.ac.in

Contents

PART

I

Task 1 - MDP Planning

Section 1

Value Iteration

Algorithm 1 Value Iteration

Input : numStates, numActions, T, R, γ
Output: V, π
$V_0 \leftarrow$ np.zeros(numStates) ; `/* Initialize array` V_0 `arbitrarily */`
repeat
  $t \leftarrow 0$
  for $s \in$ *states* **do**
    $V_{t+1}(s) \leftarrow \max_{a \in \mathrm{A}} \sum_{s' \in \mathrm{S}} T(s,a,s')(R(s,a,s') + \gamma V_t(s'))$
    $t \leftarrow t+1$
  end for
until $\|V_t - V_{t-1}\| < 10^{-12}$;
$\pi \leftarrow$ np.zeros(numStates, dtype = int); `/* calculating` π `using V */`
for $s \in$ *states* **do**
  $\pi(s) \leftarrow \mathrm{argmax}_{a \in \mathrm{A}} \sum_{s' \in \mathrm{S}} T(s,a,s')(R(s,a,s') + \gamma V(s'))$
end for

For checking the terminating condition of infinite loop, I used the condition that if absolute difference of previous value function and updated one is less than 10^{-12}, then break. After calculating the value function using vi, then I calculated the optimal policy corresponding to it as shown above.

Section 2

Howard's Policy Iteration

Algorithm 2 Howard's Policy Iteration

Input : numStates, numActions, T, R, γ
Output: V, π
$\pi \leftarrow$ np.zeros(numStates, dtype=int) ; `/* Initialize` π `arbitrarily */`
while π *has improvable states* **do**
  for **all** improvable states s, switch $\pi[s]$ to action a which has the maximum Q(s,a)
end while

During the policy evaluation step (i.e. when checking all improvable states) in the loop, Q(s,a) is calculated, and at the end, it becomes the value function corresponding to optimal policy. Here also I used precision of 10^{-12} during policy evaluation. Then, for all improvable states, the policy is updated. Note that the ties are broken arbitrarily if any two improvements are same.

Section 3

Linear Programming

Algorithm 3 Linear Programming

Input : numStates, numActions, T, R, γ
Output: V, π
Objective: Maximize$\Big(-\sum_{s \in S} V(s) \Big)$
subject to $V(s) \geq \sum_{s' \in S} T(s,a,s')(R(s,a,s') + \gamma V(s')), \forall s \in S, \forall a \in A$
LP Solver: solved using PULP_CBC_CMD solver of PuLP python module
$\pi \leftarrow$ np.zeros(numStates, dtype = int); `/* calculating π using V */`
for $s \in$ *states* **do**
  $\pi(s) \leftarrow \text{argmax}_{a \in \text{A}} \sum_{s' \in \text{S}} T(s,a,s')(R(s,a,s') + \gamma V(s'))$
end for

I encoded the objective function as shown above into the PuLP solver, by making all $V(s)$ as variables. After the solver has computed the value function, the optimal policy is calculated using the procedure shown above.

Section 4

Planner Implementation

- Initially I used 3D numpy arrays to store transition probabilites and rewards, but that was inefficient for task2, as they were very much sparse, and also the size of array was very large, taking up useless space in memory
- Therefore, I changed it and finally implemented them using list of list of dictionaries, in which first dimension is `numStates`, second is `numActions` and in third dimension, the dictionary is storing only those s' as keys, which have non-zero transition probability
- Also, I have used the pseudo-codes given in Sutton & Barto book as reference for my coding implementation of planning algorithms
- For the task1 MDPs, LP was coming out to be most efficient, then hpi and then vi. But as we proceeded to task2, vi/hpi were taking lesser time than LP, maybe due to a large number of constraints and variables.

PART

II

Task 2 - 2v1 Football

Section 5

MDP Formulation

- Number of States = $((16 * 16 * 16) * 2) + 2 = 8194$. The football field is a 4×4 grid. So, the number of positions where a player can be present are 16. Therefore there are $16 * 16 * 16$ states for different positions of B1, B2, and R. For keeping track of possession of ball between B1 & B2, this number doubles. So, total states possible in game are 8192. Now, I added 2 extra states as terminal states -
  - One for representing that the end of game with a successful goal
  - Other for representing the end of game without any goal
- Number of Actions = 10, as given in the description of 2v1 football.
- As the planner takes states in the range [0,`numStates` -1], I mapped the states of the form [B1_square, B2_square, R_square, ball_possession] to range [0,8191] in the same order as they were present in opponent policy file (i.e. first 4 bits for B1, next 4 for B2, next 4 for R, last bit for possession). Then, the two terminal states are named 8192 (successful goal end state) and 8193 (game ends without goal state)
- End States: As explained above, there are 2 end states, namely, 8192 and 8193
- MDP Type: It is **Episodic**, because there are 2 terminal states
- Discount factor: I used gamma as 1, because the game needs to be played completely until it arrives any of the terminal states, inorder to get the expected number of goals
- Transitions: For each state (except the terminals), for each action (from 0 to 9), I calculated the next state possible. Then for each action of the opponent, depending on the opponent's policy, I calculated the final state reached (s'), along with the transition probabilities, by multiplying the conditional probabilities (as described in the problem statement using parameters p and q) with opponent's probability. If the action was 9, the success probability led to state 8192 (goal), else other action's success probabilities led to the calculated s'. All the failure probabilites, as per the description in problem statement conditions, led to the terminal state of 8193 (game end without goal)
- Reward Function:

$$\text{reward} = \begin{cases} 1 & \text{if game ends with successful goal} \\ 0 & \text{otherwise} \end{cases}$$

- **Assumption**: Note that I assumed that the opponent player won't go out of bounds (after confirming with TA) and thus neglected those possibilities.

Section 6

Graphs

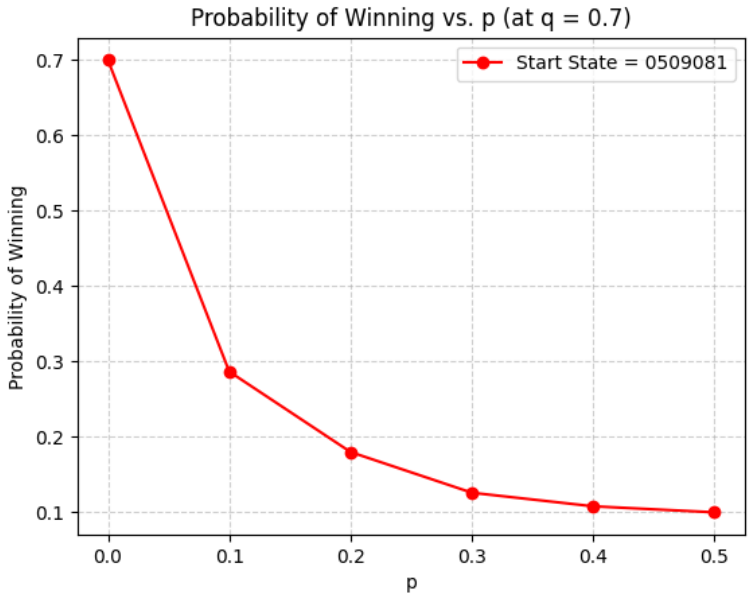

Figure 1. Probability of winning, starting from position [05, 09, 08, 1] against policy-1 (greedy defense) by varying p, keeping $q = 0.7$

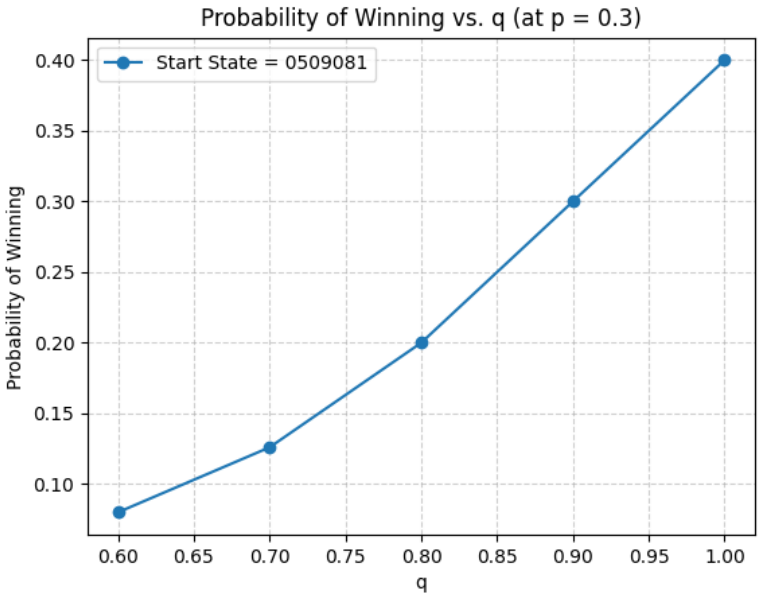

Figure 2. Probability of winning, starting from position [05, 09, 08, 1] against policy-1 (greedy defense) by varying q, keeping $p = 0.3$

Subsection 6.1
Observations

Lets look at the conditional probabilities (i.e. probabilities given an action of opponent) of those things which are in favour of our team for winning, in terms of p and q -

$$P(\text{successful movement of player with possession}) = 1 - 2 * p$$

$$P(\text{successful movement of player without possession}) = 1 - p$$

$$P(\text{successful pass}) = q - 0.1 * \max(|x_1 - x_2|, |y_1 - y_2|)$$

$$P(\text{successful goal on a shoot}) = q - 0.2 * (3 - x_1)$$

From these, some general intuitive observations are: q denotes that the attack of team is good, or team has an aggressive playing style. This is because higher q implies that the probabilities of successful passes and shots on goal are more, which emphasizes passing and shooting, leading to more scoring opportunities and, ultimately, higher win probabilities.
On the other hand, higher p will mean that there are more chances that movements of players are not successful, which may result in game ending without goal. Also, if by any means, situation of tackle occurs, then higher p will lead to successful tackle, resulting in game end without goal. From our team's point of view, we would want lower p implying that team is playing in a conservative approach, with better possession of ball.
These were my intuitions, now lets see the results obtained from graphs -

- Graph1 (For p in {0, 0.1, 0.2, 0.3, 0.4, 0.5} and q = 0.7):
  - As p increases from 0 to 0.5, the winning probability decreases from 0.7 to 0.1. I also calculated the optimal action using my python script and got them to be {1, 1, 1, 5, 5, 9} repectively.
  - This trend **matches** with my intuition, because when p is higher, then there were more chances of game ending without goal, due to chances of failures of movements and tackling, leading to low winning probability.
  - And for low values of p, the movements were likely to be successful. This combined with the optimal actions, led to higher winning probability.
  - The actions chosen are also justified. For low p, the movements are more likely to be successful, so they move. And as p becomes higher, movements may be unsuccessful, so B1 attempts goal shot.
- Graph2 (For q in {0.6, 0.7, 0.8, 0.9, 1} and p = 0.3):
  - As q increases, the winning probability also increases from 0.08 to 0.4. The corresponding optimal actions found were {1, 5, 9, 9, 9}
  - This also **matches** with the intuition, as higher q implies more & successful passes & goal attempts, giving better chances of winning with a goal.
  - And for low q, a pass or shoot may end up in a failure more likely, so the shoot attempt would not have happened much, leading to low winning chances
  - The actions also suggest same thing, as last 3 actions are of shoot, which are more likely to be successful. And when q was low, goal attempt might be unsuccessful, so the players kept moving.

Subsection 6.2
Additional file

For plotting these graphs, I made another python file named `graph.py` -

- Run it by using:

```
python graph.py
```

- I hardcoded the starting state as 0509081 and input file as `test-1.txt` in it, and I utilized the same `run()` function as given in `autograder.py` for running the encoder, planner, and decoder
- This python file on running will
  - print the winning probability (along with optimal actions) to stdout, for the starting state 0509081, for each of the (p,q) parameters
  - show (as well as save) the required graphs, named as `graph1.png` and `graph2.png`