Reinforcement Learning to Navigate Through a 6 x 6 Grid Maze

Ramkishore Rao/CS-5033

Project Outline: For the Machine Learning (ML) semester project component focusing on Reinforcement Learning (RL), I evaluated a competitive machine learning agent that navigates through a maze, represented by a 6x6 grid. The grid consisted of cells, with some cells being assigned as walls (or static obstacles) that block the path of the agent. The agent can start from any cell within the grid and can exit the grid from the top left cell or the bottom right cell, which were designated as terminal cells. Rewards were assigned as the agent traverses through the maze for visiting a cell.

Assigned rewards were as follows:

- Reward to visit a cell without an obstacle = -1
- Reward to visit a terminal cell = 0
- Reward to visit a cell with an obstacle = -1,000

When in a cell or a given state, an agent is able to take one of four actions – Up (0), Right(1), Down(2), and Left(3) to get to the next state. Note wall is assigned 5 and terminal cell is assigned 4. V(s) values for wall state are assigned a value of 13.

Problem Hypotheses: I hypothesize the following:

1. Policy iteration and value iteration algorithms will be computationally more efficient than the naïve algorithm, and will have a significantly lower value (or negative reward) relative to an equi-probable policy.
2. Q Learning and SARSA will yield a terminal state within a specified number of steps and that the negative reward within subsequent episodes will be lower than the previous ones, demonstrating that the agent is learning from previous episodes[1].
3. On policy control method (SARSA λ) will yield a terminal state and that the algorithm provides better results on Q values for intermediate values of λ[2].

Experiments: At first, a solution was developed for a maze with no obstacles and second, a solution was also developed for a maze with static obstacles, i.e., walls in it. For each type of maze, a naïve solution was developed with a equi-probable policy for the probability of changing states in any direction (Up, Right, Down, and Left). Q and SARSA learning algorithms were also utilized to evaluate whether the agent can exit the maze within a specified number of steps.

Dynamic Prgramming

Naïve Policy:

```
Input π, the policy to be evaluated, 0.25 prob for each action for each state
Threshold = 0.0001
Initialize V(s) to 0 for all states
Loop:
    Δ = 0
    Loop for each s ∈ S
        V = V(s)
        $V(s) = \sum_{s',r} p(s',r|s,\pi(s))[r + \Upsilon V(s')]$
        $\Delta = \max(\Delta, |v - V(s)|)$
until Δ < threshold
```

Policy Evaluation and Iteration

```
Threshold = 0.0001
1. Initialize V(s) to 0 for all states
2. Policy Evaluation:
Loop:
    Δ = 0
    Loop for each s ∈ S
        v = V(s)
        $V(s) = \sum_{s',r} p(s',r|s,\pi(s))[r + \Upsilon V(s')]$
        $\Delta = \max(\Delta, |v - V(s)|)$
until Δ < threshold
3. Policy Improvement
   policy_stable = True
   for each s ∈ S
        old action = $\pi(s)$
        $\pi(s) = \text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \Upsilon V(s')]$
        If old_action $\neq \pi(s)$, then $\text{policy}_{\text{stable}}$ = False
   If policy_stable, then stop and return V = v and π = π
   Else go to 2
```

Value Iteration

```
Threshold = 0.0001
Initialize V(s) to 0 for all states
Loop:
    Δ = 0
    Loop for each s ∈ S
        V = V(s)
        $V(s) = \sum_{s',r} p(s',r|s,\pi(s))[r + \Upsilon V(s')]$
        $\Delta = \max(\Delta, |v - V(s)|)$
until Δ < threshold
Output a deterministic policy π(s) = $\text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \Upsilon V(s')]$
```

No Static Obstacles Case. In this case, all three approaches noted above were evaluated. Both the value and the policy iteration outperformed the naïve policy. Results are provided on Exhibits 1 and 2.

Static Obstacles Case. I placed a static obstacle in the cells 31, 32, 33 and 34, with a reward of -1000. Policy values and final values are presented for both policy and value iteration. Values indicate that the agent is exiting from the bottom right or top left and is not entering the cells with the walls from the neighboring states (see Exhibits 4 and 5). These values were lower than from naïve policy (Exhibit 3).

Temporal-Difference (TD) Learning: TD learning refers to a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function. Three different types of TD learning algorithms were evaluated as outlined below:

Q Learning

Initialize Q(s,a) to 0 for all s ∈ S, a∈ $A(s)$, small, Ɛ >0

[1] Learning Rates for Q Learning, Eval Even Dar et al, 2003

[2] Reinforcement Learning, An Introduction, Sutton and Barto

```
Loop for each episode:
 Initialize s
   Loop for each step of episode:
     Choose a from s under policy derived from Q (Ɛ-greedy)
     Take action, observe R, s'
     $Q(s, a) = Q(s, a) + \alpha[R + \gamma \max_a Q(s', a) - Q(s, a)]$
      s = s'
   Until s is terminal
 Output a deterministic policy π(s) equivalent to:
 $\operatorname{argmax}_a Q(s, a)$
```

SARSA

```
Initialize Q(s,a) to 0 for all s ∈ S, a∈ $A(s)$, small Ɛ >0
Loop for each episode:
 Initialize s
 Ɛ, Reduce by 1/k for each episode until 10% of total episodes
 Choose a from s under policy derived from Q (Ɛ-greedy)
    Loop for each step of episode:
    Take action a, observe R, s'
    Choose a' from s' under policy derived from Q
     $Q(s, a) = Q(s, a) + \alpha[R + \gamma Q(s', a') - Q(s, a)]$
     s = s' and a = a'
     until s is terminal
 Output a deterministic policy π(s) equivalent to:
 $\operatorname{argmax}_a Q(s, a)$
```

SARSA λ-

```
Initialize Q(s,a) to 0 for all (s,a), small Ɛ >0
Loop for each episode:
Intialize e(s,a) = 0 for all (s,a); Initialize s, a
  Loop for each step of episode:
   Take action a, observe R, s'
    Choose a' from s' under policy derived from Q
    delta = R + $\gamma$ Q(s', a) – Q(s,a)
    e(s,a) = e(s,a) + 1
    for all (s,a)
          $Q(s, a) = Q(s, a) + \alpha$ delta e(s, a)
          $e(s,a) = \gamma \lambda e(s',a')$
     s = s' and a = a'
     until s is terminal
```

Results

Q Learning: The algorithm outlined above was evaluated for multiple starting states. For each starting state, the algorithm was evaluated for 200,000 episodes and 1,000 steps per episode or until terminal state was reached. At the end of the experiments (i.e., when the algorithm was implemented for all starting states), maximum value for each state action pair was estimated and the index that represents this maximum state action value was the predicted optimum action for that state. The optimum policy calculated by this method was equivalent to the policy calculated under dynamic programming (see Exhibit 6). Further the reward and the number of states averaged across starting state also tailed off with increasing episodes (see Exhibit 8 and 9), which indicates successful implementation. On Exhibits 8 and 9, the sign of the negative reward was reversed to be able to use logarithmic scale for presentation.

SARSA: SARSA provided similar results in that the agent was able to exit the grid in all the episodes experimented in almost all the simulated runs (on few runs, it did not exit the grid in less than 0.004 pct of the episodes). The optimum policy calculated by SARSA slightly varied from the policy derived by dynamic programming (see Exhibit 7). See Exhibit 8 and 9 for comparison with Q-learning.

SARSA λ: An on-policy algorithm that includes the provision for providing credit to previous states based on recency and frequency of visits using eligibility traces was also evaluated. Because the implementation is more computationally intensive, this process was only implemented for 1 starting state and for 5,000 episodes and 200 steps per episode. Intermediate values of λ yielded better results than lower and higher values of λ (see Exhibit 10).

Comparison with Related Work: Similar to the work reported by Eyal Even Dar and Yishay Mansour (2003), Q-learning converges to the optimum policy derived by DP. For SARSA λ, Root Mean Square Errors (RMSE) calculated by comparing max q(s,a) with those estimated with DP indicate that the RMSE values are lower for intermediate values of λ (0.4), which is similar to the results from the random walk experiment presented in Chapter 12 of Reinforcement Learning by Sutton and Barto.

Although the number of steps per episode tail off with increasing episodes for both SARSA and Q-Learning, the number of episodes visited by SARSA is higher than Q-learning, which can be attributed to the fact that SARSA agent may be following a less optimal than the Q-learning agent, which is similar to work for the cliff walking experiment presented in Chapter 6 of Sutton and Barto and others, including Shixiang Gu[3].

Future Work: Future work should consider the implementation of the SARSA λ experiments to full convergence to true values so that its effectiveness in achieving optimal results can be evaluated. Similar to Q-Learning and SARSA, multiple starting states should be included in the SARSA λ code to find an average by starting state to ensure consistency in final result. Instead of using static obstacles in the maze, moving obstacles that place the wall in different cells in subsequent episodes should be considered to evaluate how many more steps the agent needs to learn a more dynamic environment than the static environment evaluated in this study.

Summary: The work presented in this study compare favorably with previous related work. The agent is able to exit from the maze using both Q-Learning and SARSA algorithms and via DP. Note also fairly consistent final Q(s,a) results were obtained with Q-learning and SARSA. Optimum policy obtained via Q-Learning matches the policy obtained via DP. DP and TD-learning all outperform the naïve equi-probable policy.

[3] Continuous Deep Q-Learning with Model Based Acceleration Shixiang Gu et al, 2016

6 X 6 Grid V(s) Values

```
[[  0.   -34.   -51.92 -61.61 -66.54 -68.54]
 [-34.   -46.07 -56.15 -62.38 -65.46 -66.54]
 [-51.92 -56.15 -60.23 -62.3  -62.38 -61.61]
 [-61.61 -62.38 -62.3  -60.23 -56.15 -51.92]
 [-66.54 -65.46 -62.38 -56.15 -46.07 -34.  ]
 [-68.54 -66.54 -61.61 -51.92 -34.     0.  ]]
```

Exhibit 1: EquiProbable Policy, No Obstacles

V(s)

```
[[ 0. -1. -2. -3. -4. -5.]
 [-1. -2. -3. -4. -5. -4.]
 [-2. -3. -4. -5. -4. -3.]
 [-3. -4. -5. -4. -3. -2.]
 [-4. -5. -4. -3. -2. -1.]
 [-5. -4. -3. -2. -1.  0.]]
```

V(s)

```
[[ 0. -1. -2. -3. -4. -5.]
 [-1. -2. -3. -4. -5. -4.]
 [-2. -3. -4. -5. -4. -3.]
 [-3. -4. -5. -4. -3. -2.]
 [-4. -5. -4. -3. -2. -1.]
 [-5. -4. -3. -2. -1.  0.]]
```

Exhibit 2: V(s) for Policy/Value Iterations, No Obstacles

6 X 6 Grid V(s) Values

```
[[     0.    -2709.66  -4065.44  -4714.98  -4974.06  -5044.6 ]
 [ -3125.41  -4059.55  -4767.67  -5101.44  -5158.61  -5111.13]
 [ -5312.67  -5631.45  -5840.26  -5760.49  -5443.82  -5126.18]
 [ -7177.16  -7309.33  -7197.43  -6652.43  -5726.    -4819.59]
 [ -8905.48  -9227.27  -8983.71  -7921.79  -5984.18  -3602.59]
 [-10308.01 -10707.55 -10585.36  -9063.83  -5683.34      0.  ]]
```

Exhibit 3: EquiProbable Policy with Static Obstacles

Optimum Policy

```
[[4. 3. 3. 3. 3. 2.]
 [0. 0. 0. 0. 0. 2.]
 [0. 0. 0. 0. 1. 2.]
 [0. 0. 0. 1. 1. 2.]
 [0. 0. 1. 1. 1. 2.]
 [0. 5. 5. 5. 5. 4.]]
```

V(s)

```
[[ 0. -1. -2. -3. -4. -5.]
 [-1. -2. -3. -4. -5. -4.]
 [-2. -3. -4. -5. -4. -3.]
 [-3. -4. -5. -4. -3. -2.]
 [-4. -5. -4. -3. -2. -1.]
 [-5. 13. 13. 13. 13.  0.]]
```

Exhibit 4: Policy Iteration with Static Obstacles

Optimum Policy

```
[[4. 3. 3. 3. 3. 2.]
 [0. 0. 0. 0. 0. 2.]
 [0. 0. 0. 0. 1. 2.]
 [0. 0. 0. 1. 1. 2.]
 [0. 0. 1. 1. 1. 2.]
 [0. 5. 5. 5. 5. 4.]]
```

V(s)

```
[[ 0. -1. -2. -3. -4. -5.]
 [-1. -2. -3. -4. -5. -4.]
 [-2. -3. -4. -5. -4. -3.]
 [-3. -4. -5. -4. -3. -2.]
 [-4. -5. -4. -3. -2. -1.]
 [-5. 13. 13. 13. 13.  0.]]
```

Exhibit 5: Value Iteration with Static Obstacles

Optimum Policy

```
[[4. 3. 3. 3. 3. 2.]
 [0. 0. 0. 0. 0. 2.]
 [0. 0. 0. 0. 1. 2.]
 [0. 0. 0. 1. 1. 2.]
 [0. 0. 1. 1. 1. 2.]
 [0. 5. 5. 5. 5. 4.]]
```

Max of Q(s,a) or V(s)

```
[[ 0.  -1.  -1.9  -2.71 -3.44 -4.1 ]
 [-1.  -1.9  -2.71 -3.44 -4.1  -3.44]
 [-1.9  -2.71 -3.44 -4.1  -3.44 -2.71]
 [-2.71 -3.44 -4.1  -3.44 -2.71 -1.9 ]
 [-3.44 -4.1  -3.44 -2.71 -1.9  -1.  ]
 [-4.1  13.   13.   13.   13.    0.  ]]
```

Exhibit 6: Q-Learning with Static Obstacles

Optimum Policy

```
[[4. 3. 3. 3. 3. 3.]
 [0. 0. 0. 0. 0. 3.]
 [0. 0. 0. 0. 0. 3.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 2.]
 [0. 5. 5. 5. 5. 4.]]
```

Max of Q(s,a) or V(s)

```
[[ 0.   -1.   -1.91 -2.86 -3.84 -4.53]
 [-1.   -1.9  -2.76 -3.54 -4.24 -4.31]
 [-2.64 -2.72 -3.56 -4.65 -4.9  -5.11]
 [-3.4  -3.63 -4.32 -5.06 -5.45 -5.72]
 [-3.5  -4.21 -4.88 -7.5  -1.9  -1.  ]
 [-4.27 13.   13.   13.   13.    0.  ]]
```

Exhibit 7: SARSA with Static Obstacles

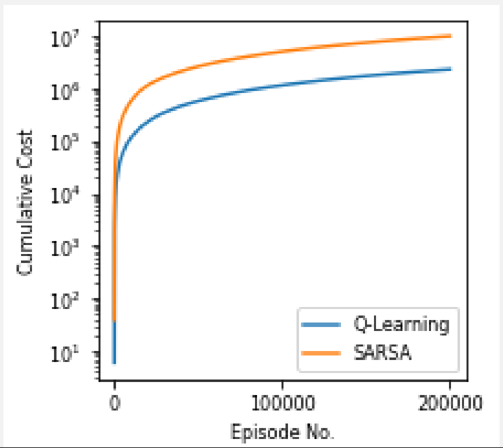

Exhibit 8: Q Learning/SARSA, Static Obstacles

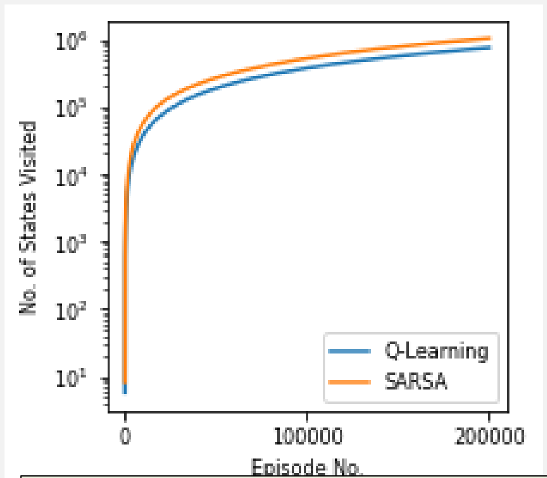

Exhibit 9: Q Learning/SARSA, Static Obstacles

λ	Learn_Rate (α)	RMSE
0.2	0.2	1.73
	0.4	1.36
0.4	0.2	0.89
	0.4	0.95
0.6	0.2	12.56
	0.4	1.89
0.8	0.2	1.17
	0.4	0.99

Exhibit 10: RMSE, SARSA λ with Static Obstacles