Samuel Singh – 1024640 **Homework 2**

MAI 5100 **2025-05-26**

Part 2

Question 1

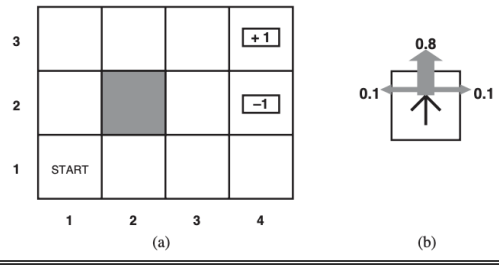


Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

1. **R&N Problem 17.1: (8 points)**
   Consider the **4x3** world shown in Figure 17.1(a). There is an impassable wall/obstacle at state (2,2). The state **(4,3)** is a terminal state with a reward of **+1**, and state **(4,1)** is a terminal state with a reward of **-1**. All other states yield a reward of **-0.04** upon exiting them.

The transition model is stochastic, as illustrated in Figure 17.1(b):

- There is an 80% chance the agent moves in the intended direction.
- There is a 10% chance the agent moves 90 degrees left relative to the intended direction.
- There is a 10% chance the agent moves 90 degrees right relative to the intended direction.

If the agent attempts to move into an external boundary or the internal wall at (2,2), it remains in its current state.

a. What are the optimal values ($V*$) of the states at (1,1), (2,1), and (1,2)?

b. What is the optimal policy ($\pi*$) for the states at (1,1), (2,1), and (1,2)?

Using value iteration, after approximately 20 iterations the values for each of the states converged when using the equation: $V(s)=\max_{s'} \sum P(s'|s,a)\cdot[R(s,a,s')+\gamma V(s')]$. Values are computed from the states closest to the terminal state and back propagated to the initial start position. As we iterate, eventually these values would converge to some final acceptable solution. ***NB. a python script was used to accelerate the iteration process.***

All values are first initialized to 0 with the exception of the two terminal states.

	Initialization			
3	0	0	0	1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

Figure 1 - Initializing all to 0

For the first iteration we inspect the state just left of the terminal state with reward 1 i.e. state (3,3). Computing the optimal values for this state yields:

- With an intention of moving right:
  - V((3,3), right) = 0.8 (-0.04 +(1 x 1)) = -0.768
  - V((3,3), up) = 0.1(-0.04 + (1x 0)) = -0.004
  - V((3,3), down) = 0.1(-0.04 + (1x 0)) = -0.004
  - Summing these gives: **0.76**

All other states will give a reward of 0 so we can compute their optimal values at this iteration in one pass eg for state (0,0):

- V((0,0), right) = 0.8 (-0.04 +(1 x 0)) = -0.032
- V((0,0), down) = 0.1 (-0.04 +(1 x 0)) = -0.004

- V((0,0), up) = 0.8 (-0.04 +(1 x 0)) = -0.004
- Summing these yield: -0.04 and can be used to represent each remaining state at this iteration.

	Iteration 1			
3	-0.04	-0.04	0.76	1
2	-0.04		-0.04	-1
1	-0.04	-0.04	-0.04	-0.04
	1	2	3	4

Figure 2 - Values after 1 iteration

For the next iteration we again start with the state (3,3) again and manually compute the optimal state values:

- With an intention of moving right:
  - V((3,3), right) = 0.8 (-0.04 +(1 x 1)) = -0.768
  - V((3,3), up but stay) = 0.1(-0.04 + (1x 0.76)) = -0.032
  - V((3,3), down) = 0.1(-0.04 + (1x-0.04)) = -0.008
  - Summing these gives: **0.832**
- With an intention of moving down:
  - V((3,3), down) = 0.8 (-0.04 +(1 x -0.04)) = -0.064
  - V((3,3), right, terminal) = 0.1 (-0.04 +(1 x -1)) = -0.104
  - V((3,3), left) = 0.1 (-0.04 +(1 x -0,04)) = -0.008
  - This sums to: -0.176
- With intention of moving left:
  - V((3,3), left) = 0.8 (-0.04 +(1 x -0.04)) = -0.064
  - V((3,3), down) = 0.1 (-0.04 +(1 x -0,04)) = -0.008
  - V((3,3), up, stay) = 0.1 (-0.04 +(1 x 0.76)) = 0.072
  - This sums to: 0.128
  -
- With intention of moving up:
  - V((3,3), up, stay) = 0.8 (-0.04 +(1 x 0.76)) = 0.576
  - V((3,3), left) = 0.1 (-0.04 +(1 x -0.04)) = -0.008
  - V((3,3), right) = 0.1 (-0.04 +(1 x 1)) = 0.096
  - This sums to: 0.664

The max of these summation is taken as: max[0.832, -0.176, 0.128,0.664] = 0.832 moving right. The optimal policy is to move right with optimal value state as 0.832. This process is repeated for all states and the results of the first four iterations are illustrated below. The values were found to converge after around 16 to 20 iterations:

	Iteration 2			
3	-0.08	-0.56	0.832	1
2	-0.08		0.464	-1
1	-0.08	-0.08	-0.08	-0.08
	1	2	3	4

Figure 3 - Values after 2 iterations

	Iteration 3			
3	0.392	0.738	0.89	1
2	-0.12		0.572	-1
1	-0.12	-0.12	0.315	-0.12
	1	2	3	4

Figure 4 - Values after 3 iterations

	Iteration 4			
3	0.577	0.819	0.906	1
2	0.25		0.629	-1
1	-0.16	0.188	0.394	0.1
	1	2	3	4

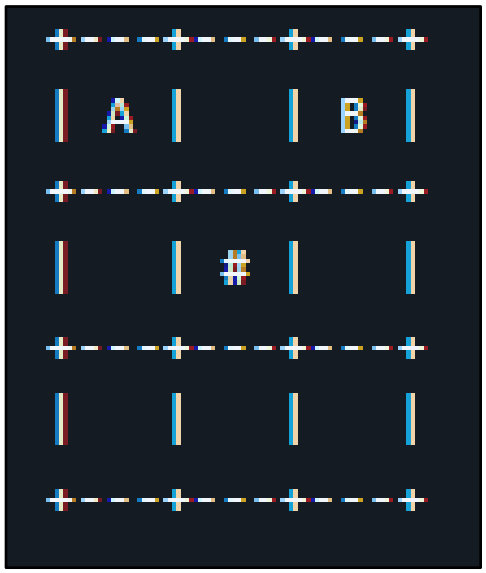
Figure 5 - Values after 4 iterations

	Iteration 20			
3	0.812	0.868	0.918	1
2	0.762		0.66	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Figure 6 - Values after 20 iterations (convergence)

Based on the values obtained from the iteration process, the optimal values and policies for states: (1,1), (2,1), and (1,2) are: [1,1] = 0.705 moving up, [2,1] = 0.655 moving left and [1,2] = 0.762 moving up.

Question 2 R&N problem



A is the starting state and B is the terminal state with reward +5. The middle square contains a wall. The agent receives a reward of –0.04 in all other states. The discount factor γ is 0.9. The agent can choose from four actions: Up, Down, Left, and Right. When the agent bumps into a wall, it stays in the same state.

a. What is the optimal value of each state (excluding B)?

b. What is the optimal policy from each state?

A similar approach to solving question one is taken where the following equation is employed: $V(s)=\max_{s'} \sum P(s'|s,a)\cdot[R(s,a,s')+\gamma V(s')]$. First all states excluding the terminal state are initialized to 0. Computation is done on the states closest to the terminal state and then values are backpropagated throughout the gird world.

	Initialization		
3	0	0	5
2	0		0
1	0	0	0
	1	2	3

Figure 7 - Initializing all to 0

For the first iteration the state (2,3) is analyzed.

- With intention of moving right:
  - V(3,3 right) = 1 (-0.04 + (0.9 x5) = 4.46
- With intention of moving left:
  - V(1,3 left) = 1 (-0.04 + (0.9 x 0) = -0.004
- With intention of moving down:
  - V(2,2 down, stay) = 1 (-0.04 + (0.9 x 0) = -0.004
- With intention of moving up:
  - Same as moving down

Taking the max of these values gives 4.46 moving right as the optimal value and policy for this state. This approach is repeated for 5 iterations until the values converged to give the optimal value and policy for all states. The values for these iterations are illustrated below:

	1		
3	-0.04	4.46	5
2	-0.04		4.46
1	-0.04	-0.04	-0.04
	1	2	3

Figure 8 - Values after 1 iteration

	2		
3	3.974	4.46	5
2	-0.076		4.46
1	-0.076	-0.076	3.974
	1	2	3

Figure 9 - Values after 2 iterations

	3		
3	3.974	4.46	5
2	3.537		4.46
1	-0.108	3.537	3.974
	1	2	3

Figure 10 - Values after 3 iterations

	4		
3	3.974	4.46	5
2	3.537		4.46
1	-0.108	3.537	3.974
	1	2	3

Figure 11 - Values after 4 iterations

For the fifth iteration and state (2,3) analyzed:

- With intention of moving right:
  - V(3,3 right) = 1 (-0.04 + (0.9 x5) = 4.46
- With intention of moving left:
  - V(1,3 left) = 1 (-0.04 + (0.9 x 3.974) = 3.5
- With intention of moving down:
  - V(2,2 down, stay) = 1 (-0.04 + (0.9 x 4.46) = 3.974
- With intention of moving up:
  - Same as moving down
- Taking max of these = 4.46 moving right.

For the fifth iteration and state (1,3) analyzed:

- With intention of moving right:
  - V(2,3 right) = 1 (-0.04 + (0.9 x4.46) = 3.9741
- With intention of moving left:
  - V(1,3 left, stay) = 1 (-0.04 + (0.9 x 3.974) = 3.9740
- With intention of moving down:

    - V(2,2 down) = 1 (-0.04 + (0.9 x 3.537) = 3.143
- With intention of moving up:
    - Same as moving left
- Taking max of these = 3.974 moving right.

For the fifth iteration and state (1,2) analyzed:

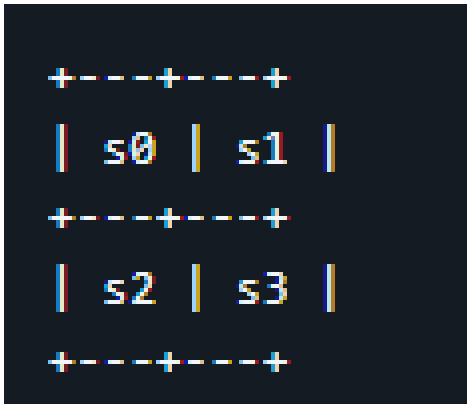
- With intention of moving right:
    - V(2,2 right, stay) = 1 (-0.04 + (0.9 x 3.537) = 3.143
- With intention of moving left:
    - Same as moving right
- With intention of moving down:
    - V (1,1 down) = 1 (-0.04 + (0.9 x -0.108) = -0.1372
- With intention of moving up:
    - V (1,3 up) = 1 (-0.04 + (0.9 x 3.974) = 3.5366
- Taking max of these = 3.537 moving up.

The same approach is used for the remaining states to give the following value/policy output:

	5		
3	3.974, Right	4.46, right	5
2	3.537, up		4.46, up
1	3.143, right/up	3.537, right	3.974, up
	1	2	3

Figure 12 - Values after 5 iterations

Question 3



State s3 is a terminal state with reward +1. All other transitions have reward 0. From each non-terminal state, the agent can move in four directions: Up, Down, Left, Right. If the agent would move off the grid, it stays in the same state. The discount factor γ is 0.8 and the learning rate α is 0.5.

a. All Q-values are initialized to 0. The agent performs the following experiences (state, action, reward, next state): (s0, Right, 0, s1), (s1, Down, 0, s3), (s3, NA, 1, Terminal), (s0, Down, 0, s2), (s2, Right, 0, s3), (s3, NA, 1, Terminal). Manually perform Q-learning updates for each of these experiences and show your calculations.

Using the formula for Q-learning: **$Q(s,a) \leftarrow Q(s,a)+\alpha[reward+\gamma \cdot a'maxQ(s',a')-Q(s,a)]$** where Q(a,s) is the old value, α is the learning rate, γ is the discount factor, reward is the reward obtained from the action and a'maxQ(s',a') is the max possible Q value for the next state.

All state action pairs are initialized to zero i.e. Q(s,a) = 0.

1. For the first experience and state s0: (s0,right,0,s1):
    - Q(s0, Right) = 0 + 0.5 [0 + 0.8 x maxQ(s1) − 0]
    - Q(s0, Right) = 0 + 0.5 [0 + 0.8 x 0 − 0]
    - Q(s0, Right) = 0

2. For the second experience (s1, Down, 0, s3)
    - Q(s1,Down) = 0 + 0.5 [0 + 0.8 x maxQ(s3) − 0]
    - Q(s1,Down) = 0 + 0.5 [0 + 0.8 x 0−0]
    - Q(s1,Down) = 0

3. For the third experience (s3, NA, 1, Terminal)
    - Q(s3,NA) = 0 + 0.5 [1 + 0.8 x 0−0]
    - Q(s3,NA) = 0.5

4. For the fourth experience (s0, Down, 0, s2)
    - Q(s0,Down) = 0 + 0.5 [0 + 0.8 x maxQ(s2) − 0]
    - Q(s0,Down) = 0 + 0.5 [0 + 0.8 x 0 − 0]
    - Q(s0,Down) = 0

5. For the fifth experience (s2, Right, 0, s3)
    - Q(s2,Right) = 0 + 0.5 [0 + 0.8 x maxQ(s3) − 0]
    - Q(s2,Right) = 0 + 0.5 [0 + 0.8 x 0.5 − 0]
    - Q(s2,Right) = 0.2

6. For the sixth experience (s3, NA, 1, Terminal):
    - Q(s3,NA) = 0.5 + 0.5 [1 + 0.8 x 0−0.5]
    - Q(s3,NA) = 0.5 + 0.5 [1−0.5]
    - Q(s3,NA) = 0.5 + 0.25
    - Q(s3,NA) = 0.75

b. After these updates, what action would an ε-greedy agent with ε = 0.2 choose in state s0? Show your work.

With an ε = 0.2 then the agent has a 20% chance of deviating from the intended action and a 80% chance of actually taking the intended action. From the previous question it was found that for the state s0, for all valid movements the Q values were 0. So for state s0 then all actions have the same Q-value. If multiple actions are then tied for the best Qvalue the tie is broken evenly. However, in this case since all Q-values are the same, there is no bias towards any specific action regardless of how the tie is broken. Since s0 the actions are up, down, left, right the probability of any action is ¼ = 0.25. This means that the agent will choose an action at random from the set of available actions.

c. What are the pros and cons of using a higher value of α (e.g., 0.9) versus a lower value (e.g., 0.1) in Q-learning?

For a higher ε values such as ε = 0.9 the agent will learn a lot faster and adapt more rapidly to newer data. This can prove to be beneficial in environments that are dynamic. With this value of ε though, decisions are made more based on recent data as opposed to older ones and thus this approach would be better suited where values of newer data are more reliable. The major downside using a higher ε is that if the environment is sufficiently dynamic it is entirely possible that the Q values never converge. Also since there is a bias towards learning from newer data, the agent might 'forget' some older, albeit, important data.

Conversely if we set the ε to be relatively low, eg 0.1 there will be longer, but a more stable convergence. This is especially beneficial if the rewards are 'noisy'. This can help to desensitize the agent to spontaneous, one -off, random events in the environment and enable the agent to have a good 'long term' memory. As described above though, with a lower ε the agent will take a lot longer to learn and it might make it difficult for the agent to adapt in environments that are highly stochastic.

Question 4

Feature-Based Q-Learning: (8 points)
Suppose you are implementing an approximate Q-learning agent for Pacman using the following features:

- $f_1(s,a) = 1$ if action a moves Pacman closer to the nearest food, 0 otherwise
- $f_2(s,a) = 1$ if action a moves Pacman away from the nearest ghost, 0 otherwise
- $f_3(s,a)$ = number of food pellets remaining in state s

a. If the weights are $w_1 = 2.0$, $w_2 = 3.0$, and $w_3 = -0.5$, calculate Q(s,a) for a state where action a moves Pacman closer to food, away from a ghost, and there are 4 food pellets remaining.

The Q(s,a) = w1f1(s,a) + w2f2(s,a) + w3f3(s,a).

Given that f1(s,a) = 1 ie. moving towards a food pellet, f2(s,a) = 1 ie. moving away from a ghost and finally f3(s,a) = 4 ie. the number of food pellets remaining and $w_1 = 2.0$, $w_2 = 3.0$, and $w_3 = -0.5$ then the Q(s,a) can be computed as:

Q(s,a) = 2.0(1) + 3.0(1) - 0.5(4)

Q(s,a) = 3

b. Suppose Pacman takes this action and arrives in a state where there are now 3 food pellets remaining. The reward for this transition is 1. Assuming γ = 0.9, calculate the temporal difference (TD) error for this transition.

The TD(temporal difference) error, δ can be calculated using the formula:

$$\delta=[r+\gamma \cdot a' \max Q(s',a')]-Q(s,a)$$

where the reward, r = 1, discount factor γ = 0.9 the new state has 3 food pellets remaining. An assumption is made that the new action state pair is a movement towards a food pellet and away from a ghost. Based on this then: f1 = 1, f2 = 1 and f3 = 3. Then:

$$Q(s',a')=2(1)+3(1)-0.5(3)=2+3-1.5=3.5$$

Then the TD can be calculated as:

$$\delta = [1+0.9(3.5)] - 3.0 = 1+3.15-3.0$$

$$\delta = 1+3.15-3.0$$

$$\delta = 1.15$$

c. Using the TD error from part b and a learning rate α = 0.2, calculate the updated weights after this experience.

Given that $\alpha = 0.2$ and $\delta = 1.15$:

a) w1 = 2.0 + (0.2 x 1.15) = 2.23
b) w2 = 3.0 + (0.2 x 1.15) = 3.23
c) w3 = -0.5 + (0.2 x 1.15 x 4) = 0.42

d. What might be a better feature than simply counting the number of food pellets? Briefly explain why.

A better feature would be to check the actual distance to the food pellet rather than just checking the amount of food pellets. An effective method that can be used is one that was utilized in previous assignments where we made use of the Manhattan / Euclidean distance. This is overall a better feature as it gives spatial information to determine how far away a reward as the current feature of just counting the remaining food pellets doesn't tell pacman if the food pellet is one step away or one hundred. This approach also creates a gradient that guides the pacman towards food more effectively as opposed to 'randomly' stumbling across a food pellet. The use of the distance to food pellet can be expanded upon and modified eg. use of the inverse of the distance value to smoothen the gradient. Another interesting use case is that the distance variable can enable development of a food density map which the pacman can use to determine the areas that are worth exploring as opposed to 'barren' areas.

Question 5

Exploration-Exploitation: (6 points)
In ε-greedy Q-learning, we've discussed how the exploration rate ε affects learning.

a. What problems might occur if ε is set too high? What if it's set too low?

If the ε is set too high, then it means the agent will be heavily biased towards choosing random actions. If the agent is acting randomly then then it means that it isn't taking advantage of learned experiences thereby making it much less efficient. Consequently, convergence towards the optimal policy will take longer and it's possible that the agent might never actually fully exploit good strategies. Conversely, if the value is set too low then the agent will only ever do what it has already learned, which has its own merits however this means that the agent would not be likely to explore new actions which could potentially lead to a more optimal policy leading to overall suboptimal performance.

b. Many implementations decrease ε over time. Why is this a good strategy? How might you implement a schedule for decreasing ε?

When the agent doesn't have a lot of experience, ie. when the agent doen't know a lot about its environment it might be beneficial to explore more and gather more information about its surroundings. As the agent acquires more experience and information, it becomes more 'confident' and can reduce exploration while increasing exploitation. A simple schedule for deceasing ε is to take a linear approach where ε = ε_initial – decay_rate * episode. This means that some initial value of ε is set (relatively high) and for each episode this value is reduced by the episode count times some decay_rate. The decay_rate here is akin to a hyper-parameter and might require trial and error to fine tune.

c. Besides ε-greedy, describe one other approach to balancing exploration and exploitation in reinforcement learning.

Another approach to ε greedy that can be adopted to balance exploration and exploitation is the Upper Confidence Bound (UBC) approach. In this approach acitons are selected based not only on their expected values but also on the uncertainty of the estimate. This means that the actions that are less frequently tried are given higher priority thereby making the agent more likely to explore them. One common implementation of this approach is as follows:

Q(a) + c * sqrt(ln(t) / N(a)) where Q(a) is the estimated value of some action a, N(a) is the number of times the action 'a' was tried, t represents the total number of steps and c is a constant that helps to control the balance of exploration-exploitation.

Submission Questions

For your final submission, you will create a pull request (PR) to the repository. Your PR should include:

1. A detailed description of the changes you made.
    - Modified the __init__method to ensure that Q-values are stored correctly.
    - Implemented the get_QValue)self,state,action)
    - Implemented computeValueFromQValues(self, state)
    - Implemented computeActionFromQValues(self, state)
    - update(self, state, action, nextState, reward).
    - Answered questions 1-5 in section 2.
    - Was unable to complete question 6 of section 1 due to time constraints.
2. Answers to the following questions:
    - What challenges did you run into?
        i. I was absent from a few of the classes that were necessary to understand the topics needed to effectively answer the questions in this assignment. As a consequence, to this I needed to spend some time catching up on the requisite content. This is time that could have otherwise been used to do the actual assignment. Initially I had some difficulties understanding how the Q-values were computed and thus I was unable to properly tackle the assignment. After some additional reading and practice manually doing practice questions I was in a better position to handle the assignment. I found that doing the section 2 first was beneficial as it helped to reinforce the theory I would have recently learned. This made it much easier to go about implementing the code. Overall it helped me to understand why the Q values start at 0 and evolve over time.
    - Which section was most fun and why?
        i. This was definitely implementing the Q learning update and watching the agent learn. It felt rewarding to apply theoretical formulae to code and

see the agent improve its behavior over time. This was especially relevant to me given my strong affinity for robotics and control systems.

- How did you approach debugging your code?
    i. I used a structued unit testing approach where I manually computed values for specific transitions and compared same with code generated values.
    ii. I also made use of the cli tools such as the –noise parameter provide for us in the instructions.
    iii. This might have been some confirmation bias but I matched the output of the code with my expected output.
- What did you learn from this assignment?

This assignement helped to reinforce my understanding of model-free reindforcement learning. Since due to my absence from call a lot of the topics were novel, this assignement forced me to learn how agents can improve their decision making over time by updating estimated values given some reward and future states. It also helped me to understand the importance of balancing exploration with exploitation and how each would be beneficial under different circumstances (question 5). Overall, it helped to improve my confidence in translating the theory of reinforcement learning into actual code.