Reinforcement Learning

Assignment #3 Report

Student names
Pedram Abdolahi Darestani
Zahra Jabari

Student IDs
202383919
202291677

Date
August 10th, 2024

Part 1

This part is aimed at finding the optimal policy for a customized 5x5 gridworld problem using Sarsa and Q-learning. In order to address the states in the gridworld problem, we make use of the numbering convention shown below.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

For example, the blue square is in position 20, the terminal states are in positions 0 and 4, the red wall states are in positions 10, 11, 13, and 14, and the first and last positions are numbered as 0 and 24.

Problem specifications

- Agent starts at the blue square (position 20)
- Any action between any two squares yields a reward of -1
- If the agent enters the red squares, it receives a reward of -20 and ends up in the original position (position 20)
- An attempt to leave the grid yields a reward of -1 and does not change the state of the agent.
- An epsilon-greedy approach is selected.

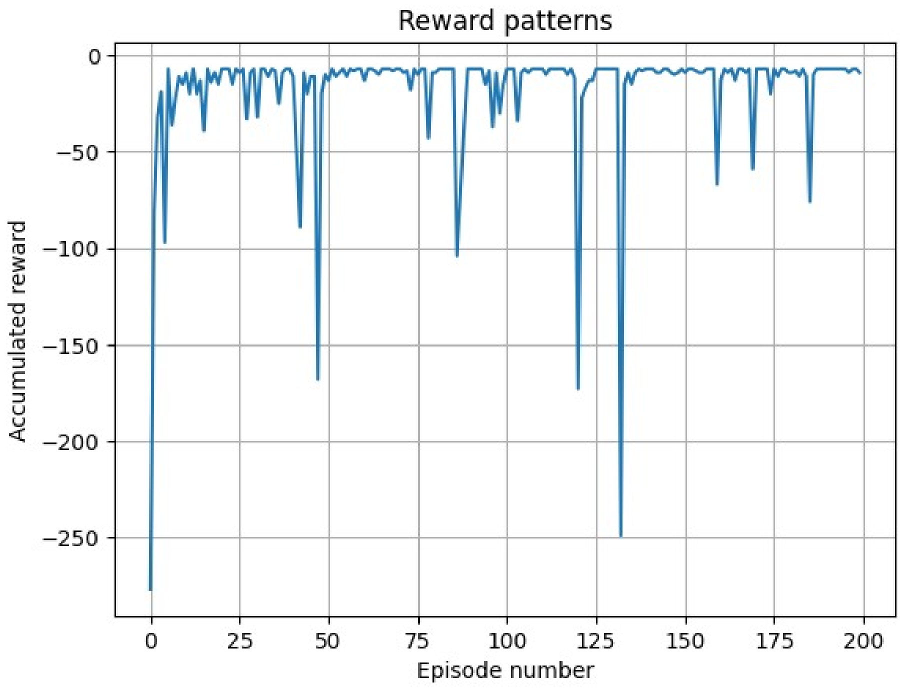
1-1) Sarsa

This algorithm was tested with different values of alpha = 0.1, 0.2, 0.3, 0.5, 0.6, and 0.7, epsilon = 0.1, 0.2, and 0.3, and discount = 0.7 to 1. Many of the runs of this algorithm with different hyper parameters were able to find a policy that leads the agent from the starting position to a terminal position. One of the found policies is shown below and the trajectory of the agent, starting from the blue square and ending in a terminal state is shown with the color green in the grid below.

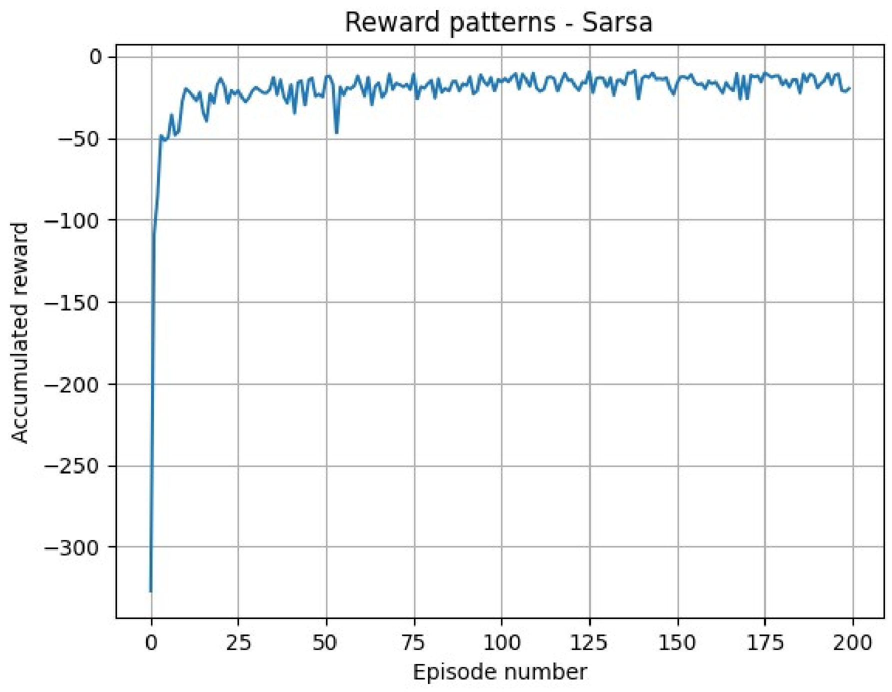
0	Left	Left	Right	4
Up	Up	Left	Up	Up
🟥	🟥	Up	🟥	🟥
Right	Right	Up	Left	Right
Right	Right	Up	Left	Left

The policy is really close to the optimal policy and the only difference it has with it is in the action in position 19. Given we have the initial position of the agent and the near

optimal policy, the agent will never make it to the state 19 for this deviation from optimality to prevent the agent from making it to the terminal state.
This algorithm was run for 200 different episodes and the sum of the accumulated rewards during each episode is plotted in the figure below.



It can be seen that the algorithm converges to the highest possible negative rewards in less than 50 episodes. Running the Sarsa algorithm with other hyper-parameters yielded similar patterns, however, quite a bit of variation was seen between different runs of the algorithm with the same set of hyper-parameters. In order to evaluate the performance of the algorithm, we reduce the effects of this type of variation through running the algorithm with one setting for 100 times (train a unique model for 200 times in each of those 100 runs) and average the accumulated rewards over episode numbers. The resulting aggregate reward pattern is shown below.



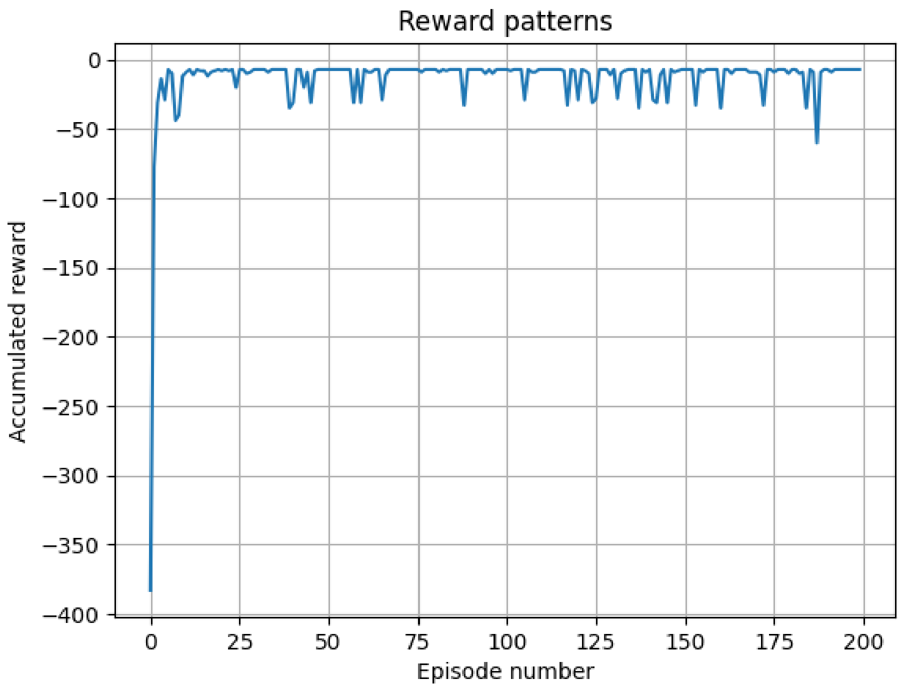
As it can be seen, the spikes in the single run of Sarsa are dampened through the averaging the results of 100 different runs and on average, the Sarsa algorithm converges in about 50 episodes.

1-2) Q-learning

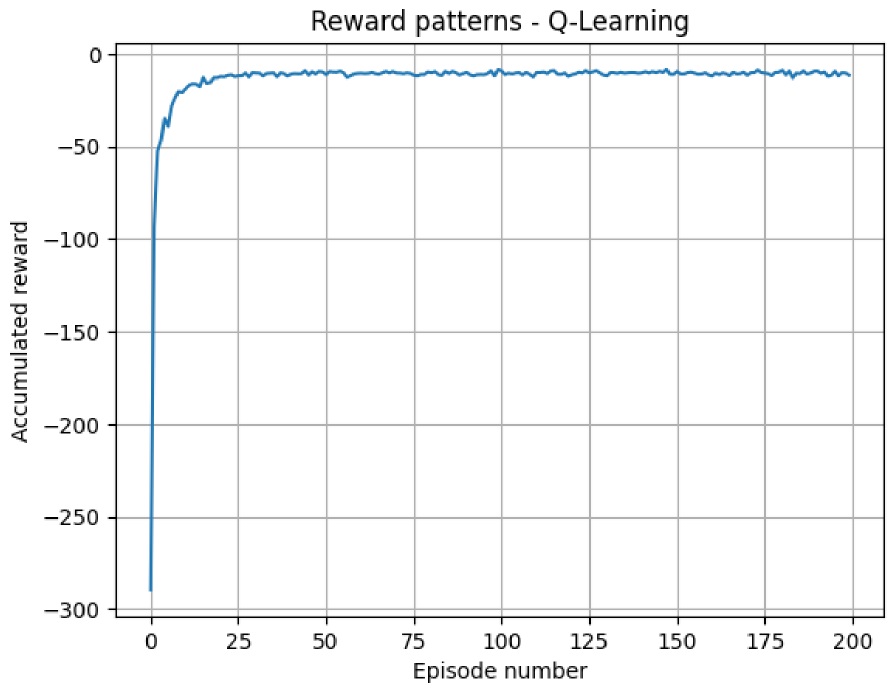
This algorithm was tested with different values of alpha = 0.1, 0.2, 0.3, 0.5, 0.6, and 0.7, epsilon = 0.1, 0.2, and 0.3, and discount = 0.7 to 1. For this algorithm, too, many of the runs with different hyper parameters were able to find a policy that leads the agent from the starting position to a terminal position. One of the found policies is shown below and the trajectory of the agent, starting from the blue square and ending in a terminal state is shown with the color green in the grid below.

0	Left	Left	Right	4
Up	Up	Left	Right	Up
🟥	🟥	Up	🟥	🟥
Right	Right	Up	Left	Left
Right	Right	Up	Up	Down

The results of the Q-learning algorithm are quite close to the Sarsa: the agent finds the shortest path to one of the terminal states and never makes it to states in which deviation from optimality can prevent the agent from reaching a terminal state or slow it down in this task.
This algorithm was run for 200 different episodes and the sum of the accumulated rewards during each episode is plotted in the figure below.



It can be seen that the algorithm converges to the highest possible negative rewards in about 15 episodes, which is much better than the results of Sarsa. Running the Q-learning algorithm with other hyper-parameters yielded similar patterns as well, but with variations in different runs. We carry out the said aggregation method for this algorithm as well and arrive at the following plot.



Similar to Sarsa, the spikes in the single run of Q-learning are dampened through the averaging the results over 100 different runs. We can see that the single instance run of the Q-learning algorithm converges in about 15 episodes while the averaged version of the accumulated rewards over 100 different runs show that this method converges in about 25 episodes. This means that the singular run was a better than average instance of the possible outputs of the algorithm.

Analysis

The results from both Sarsa and Q-learning algorithm are quite similar in the sense of their capacity in finding a policy that leads the agent to a terminal state in the least number of steps while achieving the highest possible rewards. The reason for this is that both algorithms attempt to solve the same problem with different degrees of exploration, but given enough time to train they both can find an optimal trajectory for the agent. However, by looking at the average performance of the two algorithms over 100 independent runs, it is clear that the Q-learning algorithm can find the (near) optimal policy in half as many episodes (in 25 episodes as compared to the 50 episodes in Sarsa) as Sarsa can. Moreover, Q-learning shows much less variations and spikes in its path to convergence as compared to Sarsa. The reason for this could be that Q-learning is an off-policy algorithm which results in faster convergence speeds as compared to the on-policy algorithms like Sarsa. As a result, we can say that on average Q-learning is the better algorithm for this task.

If you would like to see the results and plots for all of the hyper-parameter experimentations, you can refer to the Runs1.ipynb file available in the repository.

Part 2

This Part is aimed at evaluating a given policy for a customized 7x7 gridworld problem and estimating its state value function. In order to address the states in the gridworld problem, we make use of the numbering convention below.

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48

The blue square is in position 24, the terminal states are in positions 6 and 42, and the first and last positions are numbered as 0 and 48.

Problem specifications

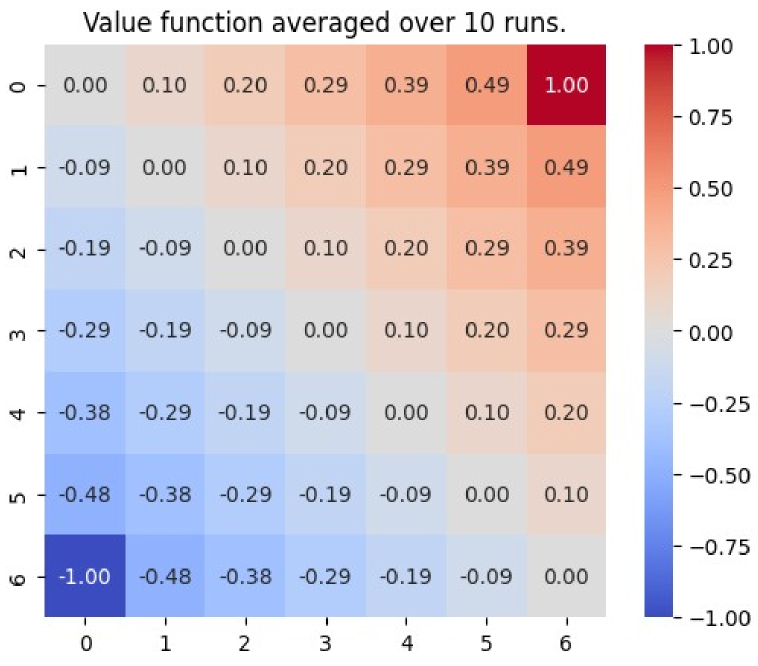
- Agent starts at the blue square (position 24)
- Any action between the white squares yields a reward of 0
- An attempt to leave the grid yields a reward of 0 and does not change the state of the agent.
- State 6 and 42 are terminal states and yield rewards of 1 and -1 respectively.
- A linear mapping function between features and weights is implemented.
- The length of the **w** vector is determined by the feature vector length.
- Three feature vectors of different lengths are defined for the states.
    - Length = 2: features are the Manhattan distance between each state and the terminal states (6 and 42)
    - Length = 8: features are: i,j index of each state, square of the x and y distances of each sate from the terminal state, Euclidean distance from each state to the terminal states
    - Length = 10: same features as length = 8 with the addition of Manhattan distances of length = 2
- The feature vectors are normalized based on the maximum value of each feature in an attempt to fight divergence.

Given the nature of the problem, the (absolute value of the) value functions should be symmetrical with respect to both diagonal axes of the square. Moreover, for this problem the discount is set to 1, since there are no immediate rewards: there is only one reward in each episode and that is given at its end.

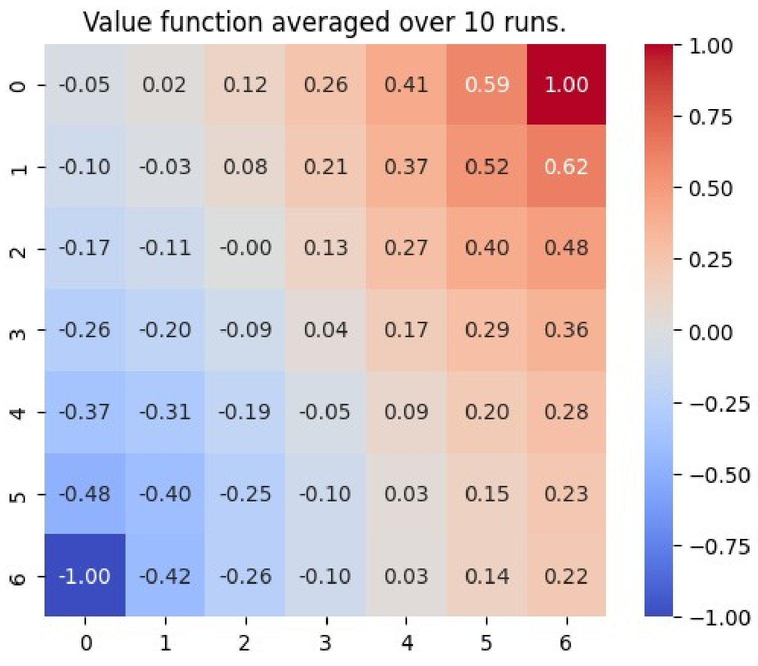
2-1) Gradient Monte Carlo prediction

The problem was solved with different values of alpha = 0.1, 0.2, 0.3, 0.4, and 0.5. There is quite a lot of variation between different runs, which is perhaps due to the random initialization of the **w** vector. As a result, we run each each hyper-parameter modification of the algorithm 10 times and then average their estimated value functions before reporting them. This can help reduce the effects of randomization variation on the performance of the model. We repeat this evaluation based on different feature vectors with various hyper-parameters and present their best estimated value functions below.

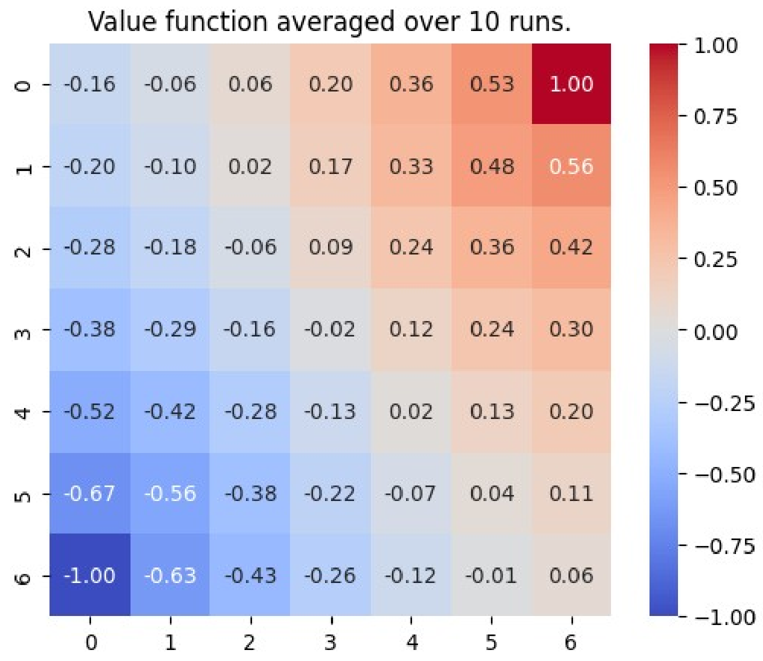
Feature vector length = 2



Feature vector length = 8



Feature vector length = 10

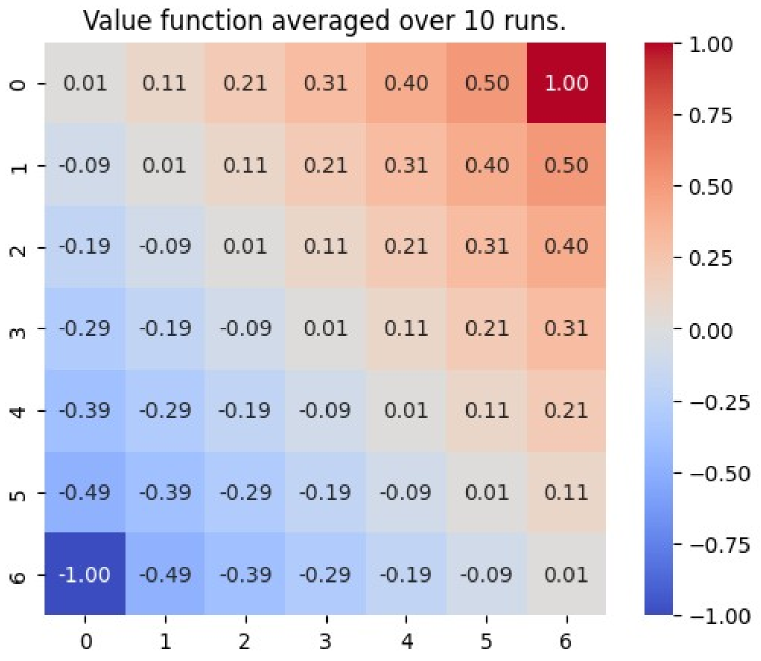


All three types of feature vectors were able to approximate the value functions corresponding to the given policy (random walk) to some extent. They all assigned higher values to the states with less Manhattan distance to the better terminal state (top right or state #6) and lower values to the ones closer to the worse terminal state (bottom left or state #42). However, of all three feature vectors only the one with length = 2 was able to achieve an almost perfect symmetry with respect to the diagonal axes. In comparison, the values of the other two value functions are closer to the ground truth (displayed in the analysis section of this question, near the end of the report).
Between the value functions found by feature vector length of 8 and 10, the one found by length = 10 is the better version. This is due to the fact that the extreme values of the value function are closer to the ground truth in the len(w)=10 version, as well as yielding a more consistent decrease and increase of value function when getting closer to the terminal states.

2-2) Semi-gradient TD(0) prediction

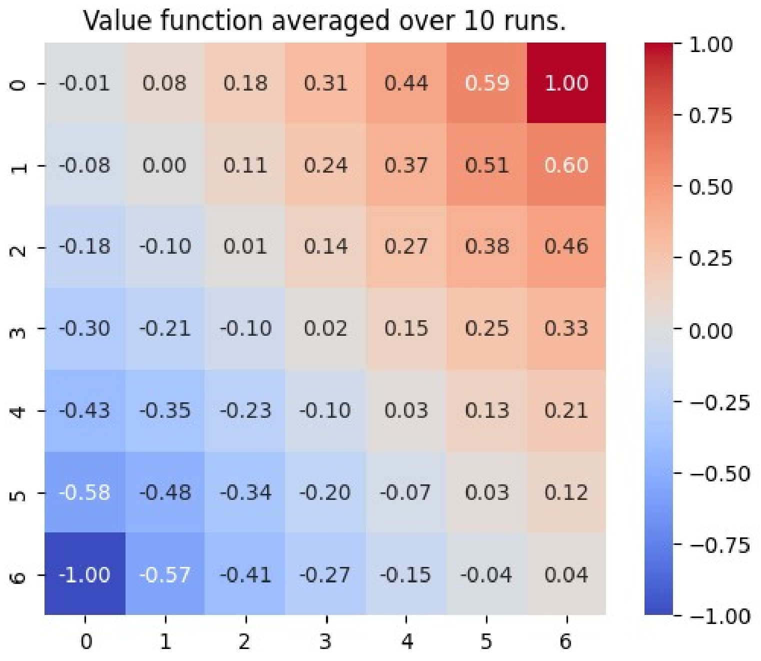
We follow the same scheme described in gradient Monte Carlo method to run the semi-gradient TD(0) algorithm and achieve the following results.

Feature vector length = 2



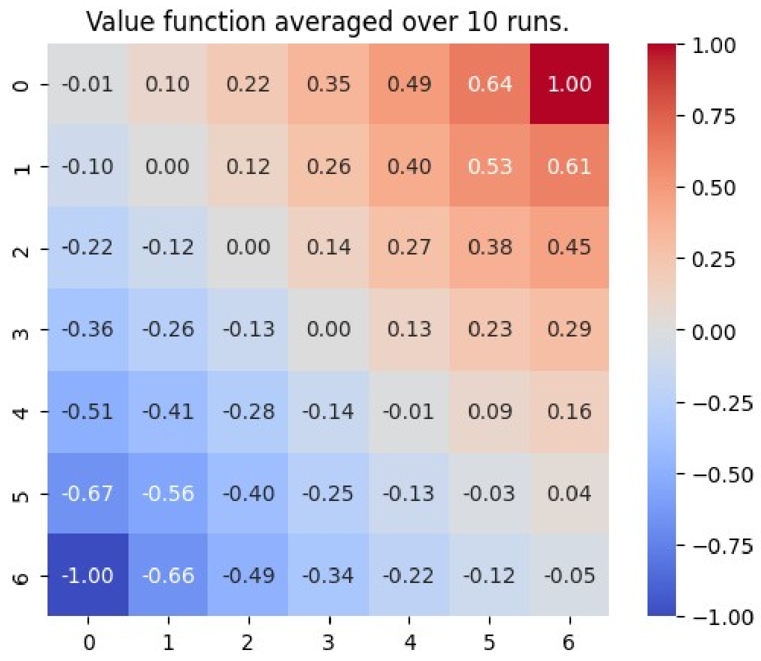
	0	1	2	3	4	5	6
0	0.01	0.11	0.21	0.31	0.40	0.50	1.00
1	-0.09	0.01	0.11	0.21	0.31	0.40	0.50
2	-0.19	-0.09	0.01	0.11	0.21	0.31	0.40
3	-0.29	-0.19	-0.09	0.01	0.11	0.21	0.31
4	-0.39	-0.29	-0.19	-0.09	0.01	0.11	0.21
5	-0.49	-0.39	-0.29	-0.19	-0.09	0.01	0.11
6	-1.00	-0.49	-0.39	-0.29	-0.19	-0.09	0.01

Feature vector length = 8



	0	1	2	3	4	5	6
0	-0.01	0.08	0.18	0.31	0.44	0.59	1.00
1	-0.08	0.00	0.11	0.24	0.37	0.51	0.60
2	-0.18	-0.10	0.01	0.14	0.27	0.38	0.46
3	-0.30	-0.21	-0.10	0.02	0.15	0.25	0.33
4	-0.43	-0.35	-0.23	-0.10	0.03	0.13	0.21
5	-0.58	-0.48	-0.34	-0.20	-0.07	0.03	0.12
6	-1.00	-0.57	-0.41	-0.27	-0.15	-0.04	0.04

Feature vector length = 10

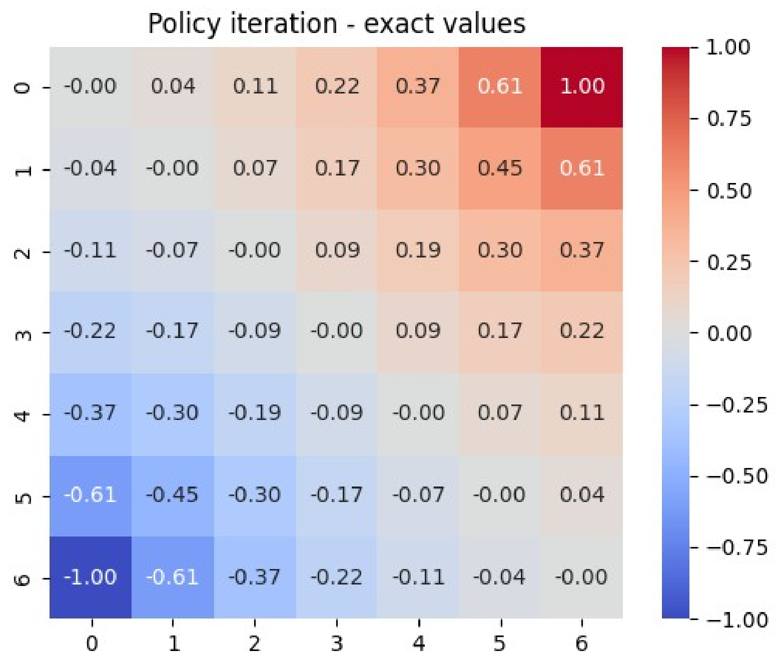


	0	1	2	3	4	5	6
0	-0.01	0.10	0.22	0.35	0.49	0.64	1.00
1	-0.10	0.00	0.12	0.26	0.40	0.53	0.61
2	-0.22	-0.12	0.00	0.14	0.27	0.38	0.45
3	-0.36	-0.26	-0.13	0.00	0.13	0.23	0.29
4	-0.51	-0.41	-0.28	-0.14	-0.01	0.09	0.16
5	-0.67	-0.56	-0.40	-0.25	-0.13	-0.03	0.04
6	-1.00	-0.66	-0.49	-0.34	-0.22	-0.12	-0.05

Similar to the previous method, by using semi-gradient TD(0) all three types of feature vectors were able to approximate the value functions corresponding to the given policy (random walk) to some extent.
Just like the previous section, only the version with len(w) = 2 was able to provide an almost perfect symmetry with respect to the two diagonal axes, while its values are farther from the ground truth than the other two.
Between the two value functions found by len(w) = 8 and len(w) = 10, it can be said that the one with len(w) = 8 is the better version. This is because the values given by this version does not exceed the ground truth at extreme points (points near the terminal states) like the model with len(w) = 10 does. Furthermore, the values of the one with length of 8 are slightly closer to the ground truth as compared to the other version.

Analysis

In order to find the exact value function of the policy, we made use of the policy iteration method and illustrated the results below.



As it can be seen, the found value function conforms to the symmetry of the problem with respect of both diagonal axes. Moreover, the value function of the states on the diagonal axis passing through the top left and bottom right are all identical and equal to 0. This is understandable since the probability of an agent getting the 1 or -1 reward while in this state by following the random walk policy is equal.
If you would like to see the results and plots for all of the hyper-parameter experimentations, you can refer to the Runs2.ipynb file available in the repository.