Ezgi Sena Karabacak
2375178

26.05.2024

EE449 Computational Intelligence

Homework 3 Reinforcement Learning

1. Basic Questions

- **Agent:** In reinforcement learning (RL), the agent is the decision-maker who interacts with the environment, taking actions based on observations to maximize cumulative reward. It learns from the consequences of its actions, adapting its strategy over time. In supervised learning (SL), the closest equivalent is the model, which makes predictions based on input data.
- **Environment:** The environment in RL includes everything outside the agent that responds to the agent's actions and provides new states and rewards. It serves as the external system with which the agent interacts, offering feedback that guides the agent's learning. In SL, this role is somewhat analogous to the dataset, which supplies the inputs and outputs for training the model.
- **Reward:** A reward in RL is a scalar value received by the agent to indicate the immediate benefit or cost of an action taken in a particular state. It serves as a feedback mechanism to help the agent evaluate the success of its actions and adjust its strategy accordingly. In SL, the loss function serves a similar purpose by providing a measure of prediction error to guide model training.
- **Policy:** In RL, a policy is a strategy or mapping from states to actions that define the agent’s behavior. It determines an agent's actions given the current state, aiming to maximize cumulative reward. This can be represented in SL as the learned function or model, which maps inputs to outputs based on training data.
- **Exploration:** Exploration in RL is trying out new actions to discover their effects and improve the agent's environmental knowledge. This approach helps the agent gather information and avoid premature convergence to suboptimal strategies. In SL, this concept is somewhat like hyperparameter tuning or model experimentation, where different configurations are tested to find the best-performing model.
- **Exploitation:** Exploitation in RL involves selecting actions that yield high rewards based on the agent's current knowledge. It focuses on using what the agent has already learned to maximize reward in the short term. In SL, this is comparable to the model's prediction or inference phase, where the model applies learned patterns to make decisions or predictions based on new data.

2. Plots

Based on the results given in Figures 1-78, the analysis of the effect of each hyperparameter and the values that produced the best results for both TD(0) Learning and Q Learning are provided below.

Learning Rate (α)

Effect:

The learning rate determines how much new information overrides old information. A high learning rate allows for faster learning but can lead to instability, while a low learning rate results in slower, more stable learning.

Best Value:

0.1: The experiments indicate that a learning rate 0.1 produced the best balance between learning speed and stability. Higher values like 0.5 and 1.0 led to instability, while lower values like 0.001 and 0.01 resulted in slower convergence.

Discount Factor (γ)

Effect:

The discount factor determines the importance of future rewards. A high discount factor values future rewards more, promoting long-term strategies, while a low discount factor focuses on immediate rewards.

Best Value:

0.95: This value consistently yielded the best results, allowing the agent to effectively consider long-term rewards, which is crucial in maze navigation.

Initial Exploration Rate (ε)

Effect:

The exploration rate controls the balance between exploration (trying new actions) and exploitation (using known actions). High exploration rates encourage discovering new strategies, while low rates focus on optimizing known strategies.

Best Value:

0.2: An exploration rate of 0.2 was found to be the most effective, providing a good balance between exploration and exploitation. Values of 0.5 and higher led to excessive exploration and slower convergence.

Detailed Results

TD(0) Learning:

Figures 1-3: With $\alpha = 0.1$, $\gamma = 0.95$, and $\varepsilon = 0.2$, the policy plots, value function plots, and convergence plots indicate stable and effective learning.

Figures 7-9, 13-15, 19-21, 25-27: Lower learning rates like 0.001 and 0.01 resulted in slower convergence. Higher rates like 0.5 and 1.0 caused instability.

Figures 31-33, 37-39, 43-45, 49-51: Discount factors lower than 0.95 led to suboptimal long-term planning.

Figures 55-57, 61-63, 67-69, 73-75: Exploration rates of 0.0 and 1.0 were less effective than 0.2.

Q Learning:

Figures 4-6: With $\alpha = 0.1$, $\gamma = 0.95$, and $\varepsilon = 0.2$, Q Learning also showed stable and effective learning.

Figures 10-12, 16-18, 22-24, 28-30: Similar to TD(0) Learning, lower learning rates led to slower convergence and higher rates caused instability.

Figures 34-36, 40-42, 46-48, 52-54: Discount factors lower than 0.95 were less effective.

Figures 58-60, 64-66, 70-72, 76-78: Higher exploration rates led to excessive exploration and were less effective than 0.2.

Summary:

Learning Rate (α): Best at 0.1.

Discount Factor (γ): Best at 0.95.

Initial Exploration Rate (ε): Best at 0.2.

These values were consistently effective across different experiments for both TD(0) Learning and Q Learning, ensuring a good balance between exploration, learning speed, and stability.

Ezgi Sena Karabacak
2375178

26.05.2024

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$)

TD Learning

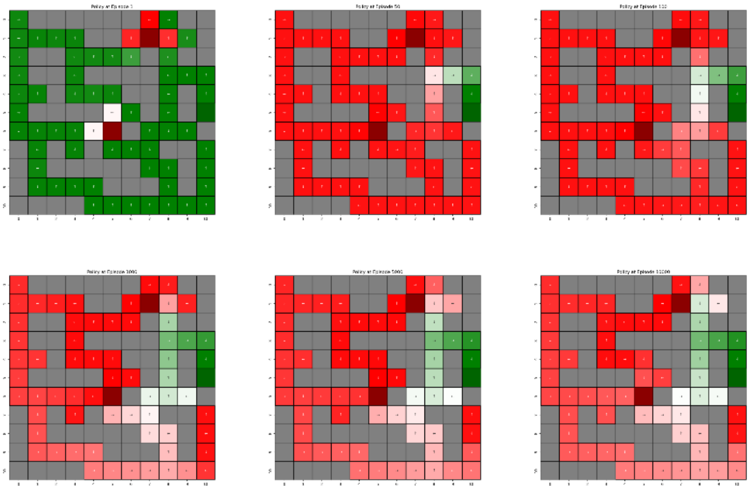

Figure 1. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$

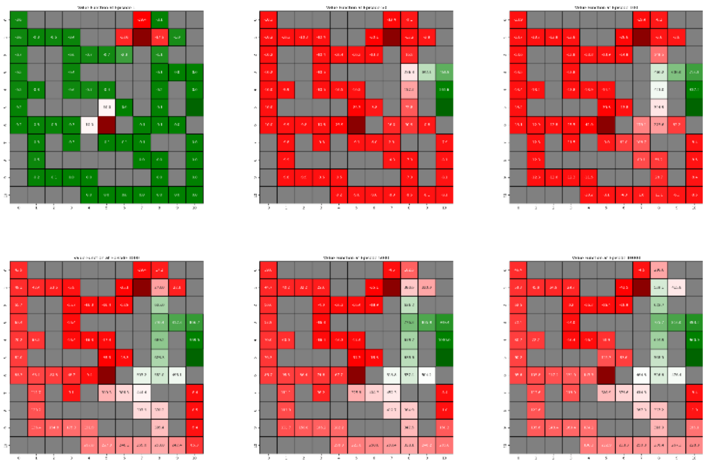

Figure 2. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$

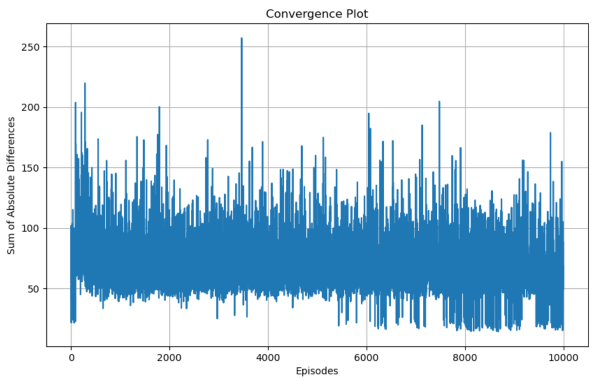

Figure 3. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$

Ezgi Sena Karabacak 26.05.2024
2375178

Q Learning

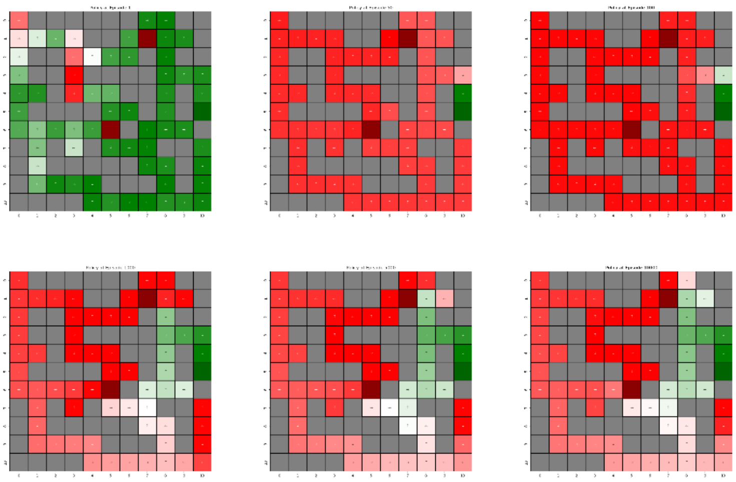

Figure 4. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$

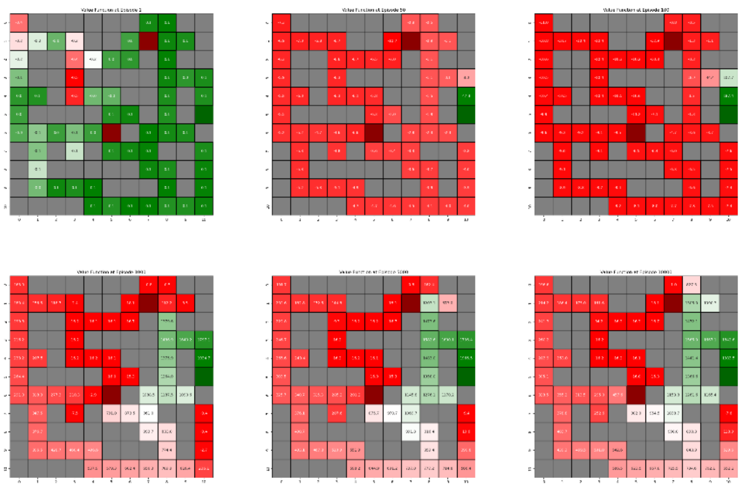

Figure 5. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$

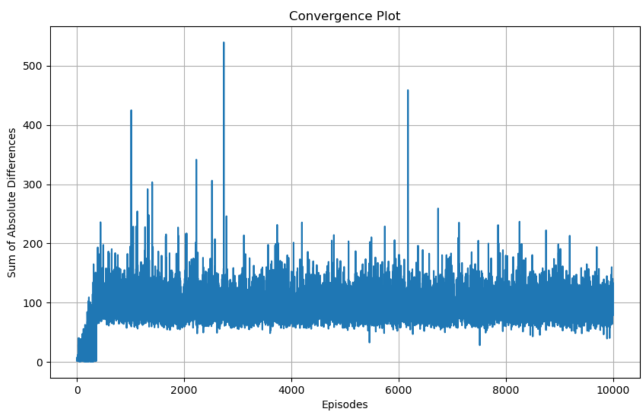

Figure 6. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.2$

Hyperparameter ($\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$)

TD Learning

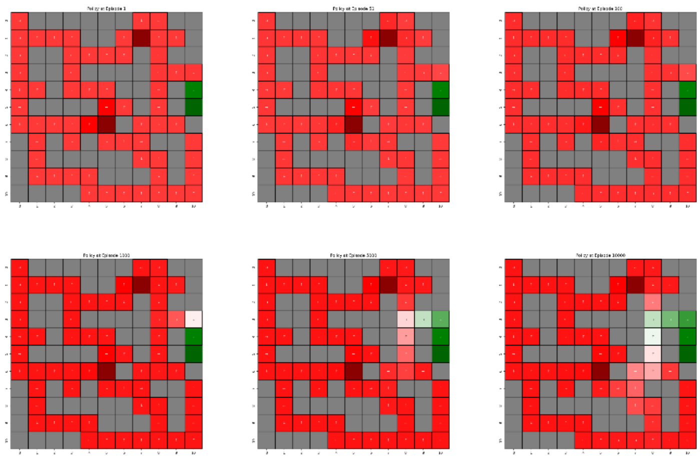

Figure 7. Policy plots for TD Learning when $\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$

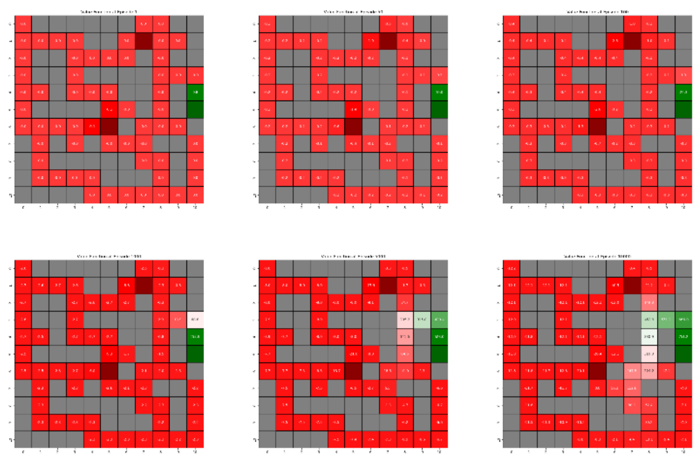

Figure 8. Value Function plots for TD Learning when $\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$

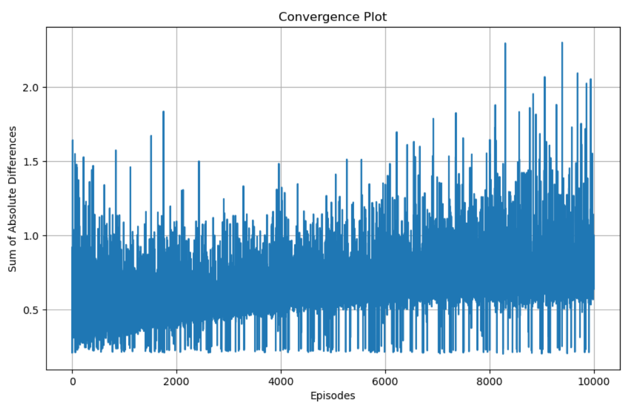

Figure 9. Convergence Plot for TD Learning when $\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$

Ezgi Sena Karabacak 26.05.2024
2375178

Q Learning

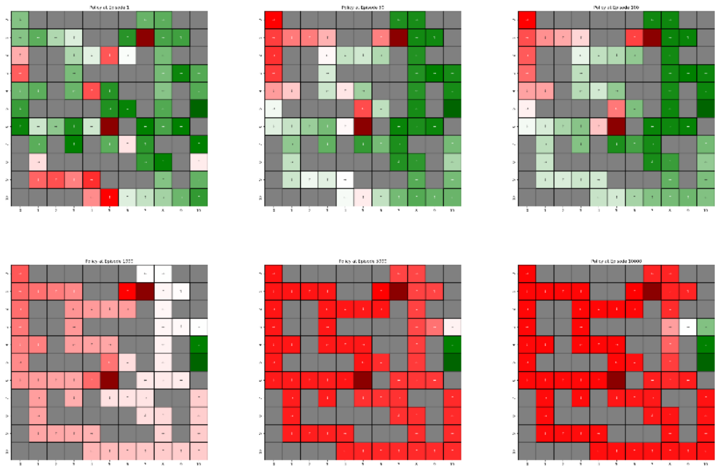

Figure 10. Policy plots for Q Learning when $\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$

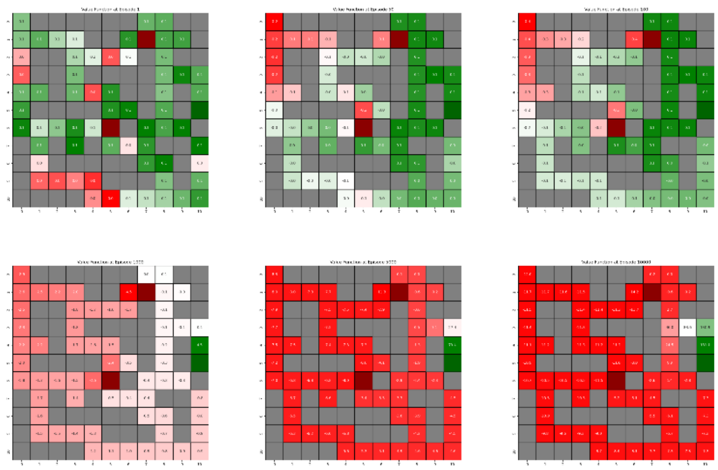

Figure 11. Value Function plots for Q Learning when $\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$

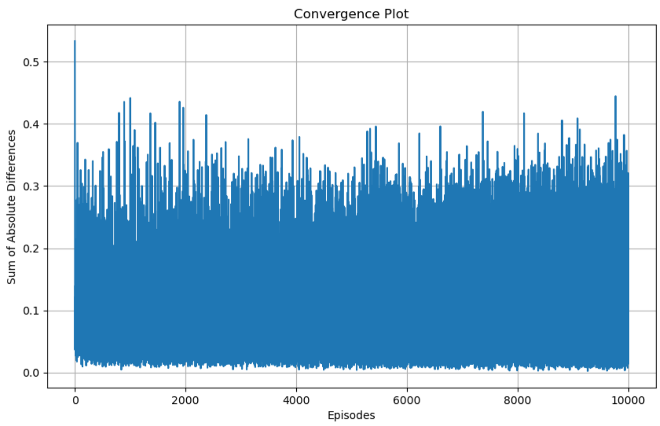

Figure 12. Convergence Plot for Q Learning when $\alpha = 0.001$, $\gamma = 0.95$, $\varepsilon = 0.2$

Hyperparameter ($\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$)

TD Learning

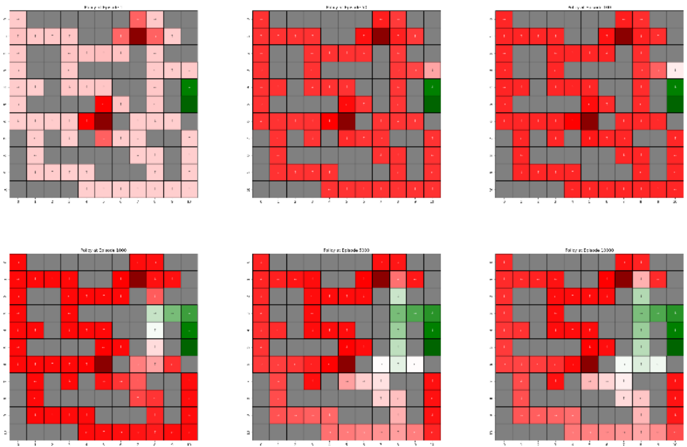

Figure 13. Policy plots for TD Learning when $\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$

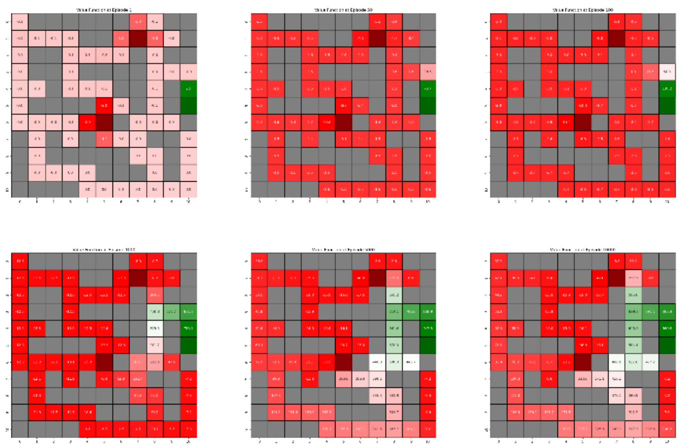

Figure 14. Value Function plots for TD Learning when $\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$

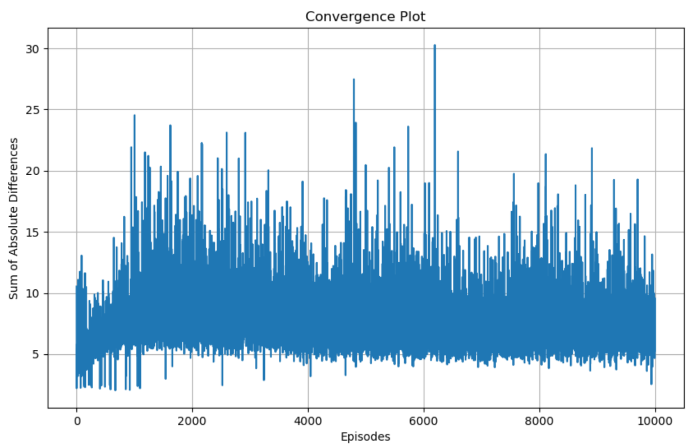

Figure 15. Convergence Plot for TD Learning when $\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$

Ezgi Sena Karabacak 26.05.2024
2375178

Q Learning

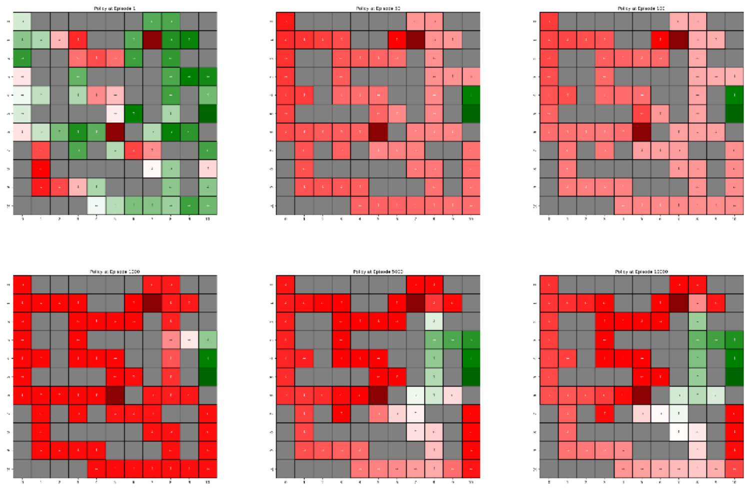

Figure 16. Policy plots for Q Learning when $\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$

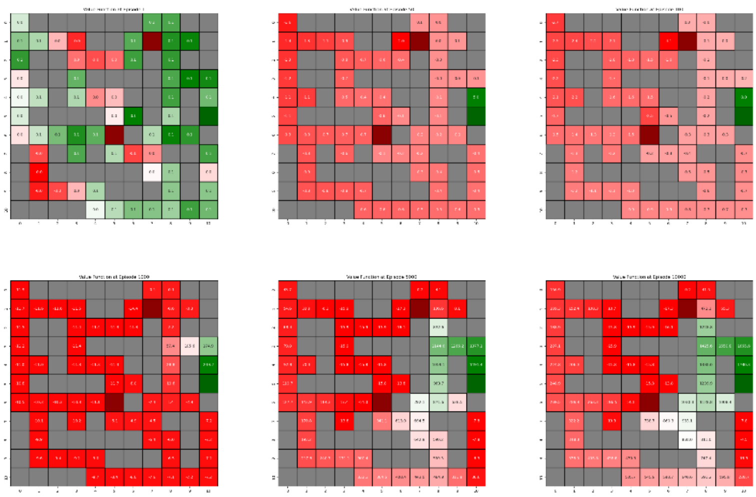

Figure 17. Value Function plots for Q Learning when $\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$

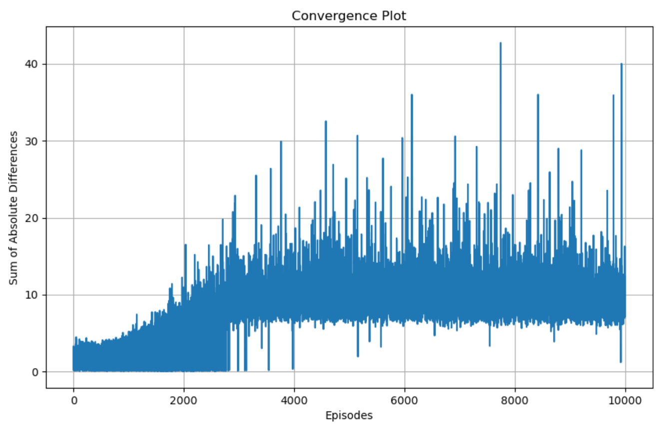

Figure 18. Convergence Plot for Q Learning when $\alpha = 0.01$, $\gamma = 0.95$, $\varepsilon = 0.2$

Hyperparameter ($\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$)

TD Learning

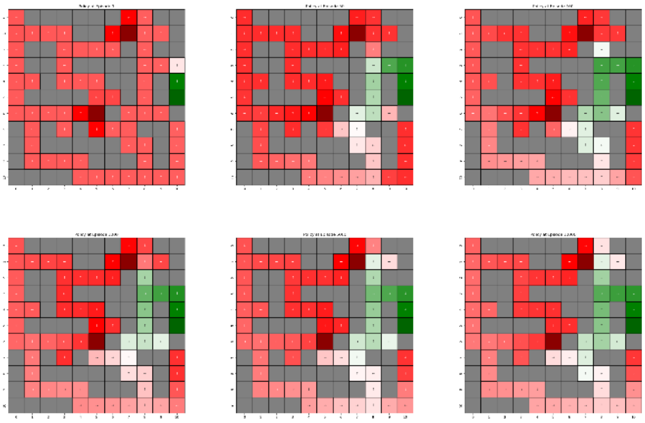

Figure 19. Policy plots for TD Learning when $\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$

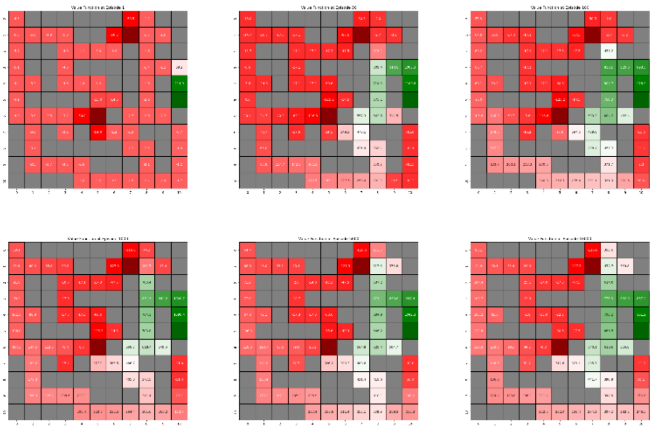

Figure 20. Value Function plots for TD Learning when $\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$

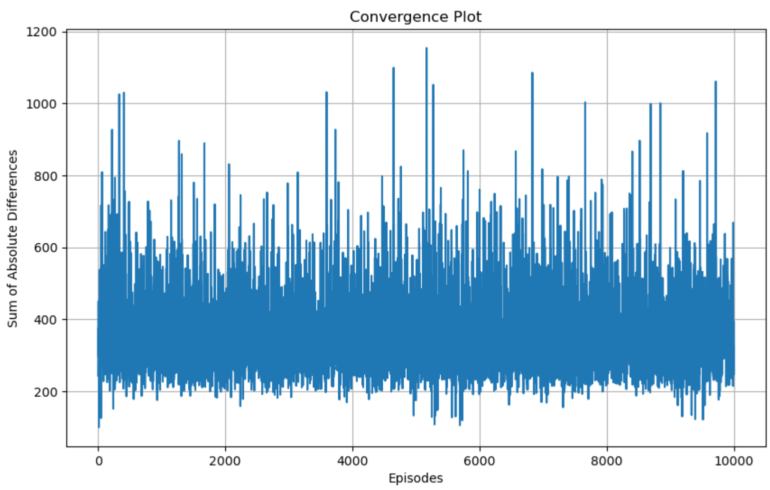

Figure 21. Convergence Plot for TD Learning when $\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$

Ezgi Sena Karabacak
2375178 26.05.2024

Q Learning

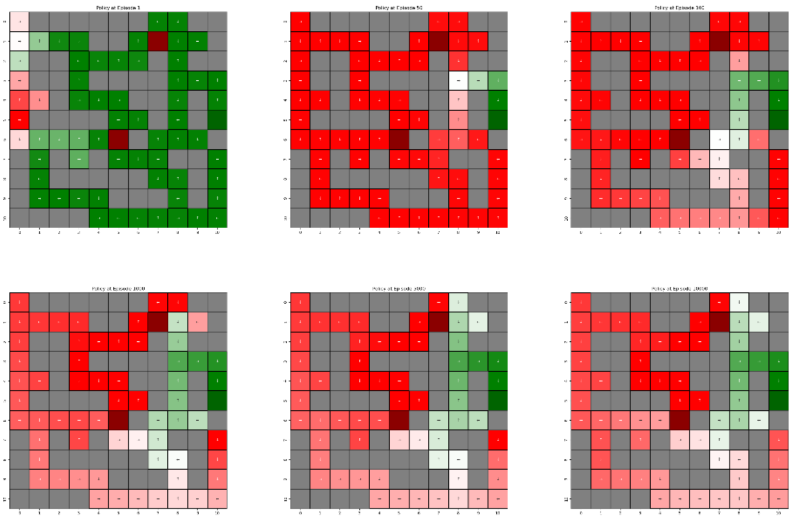

Figure 22. Policy plots for Q Learning when $\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$

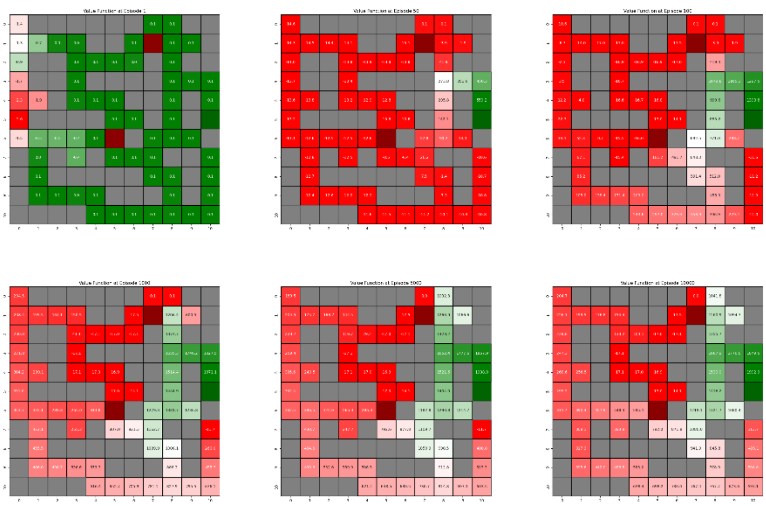

Figure 23. Value Function plots for Q Learning when $\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$

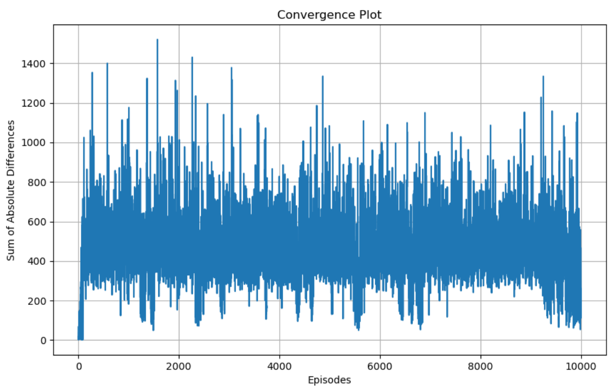

Figure 24. Convergence Plot for Q Learning when $\alpha = 0.5$, $\gamma = 0.95$, $\varepsilon = 0.2$

Ezgi Sena Karabacak
2375178

26.05.2024

Hyperparameter ($\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$)

TD Learning

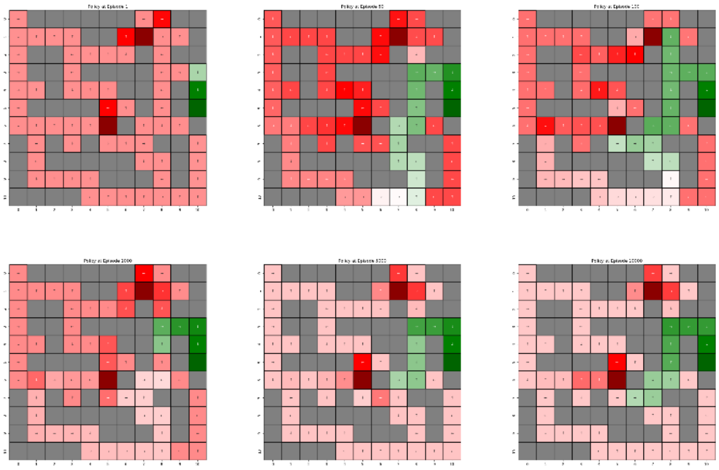

Figure 25. Policy plots for TD Learning when $\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$

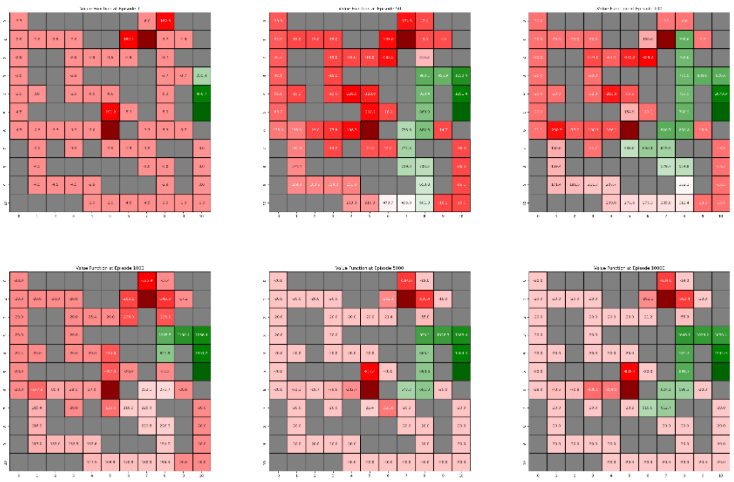

Figure 26. Value Function plots for TD Learning when $\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$

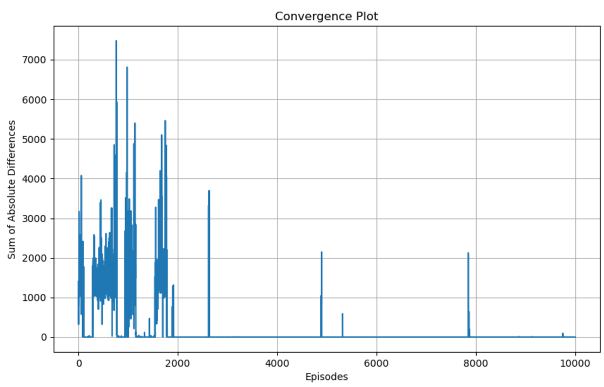

Figure 27. Convergence Plot for TD Learning when $\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$

Q Learning

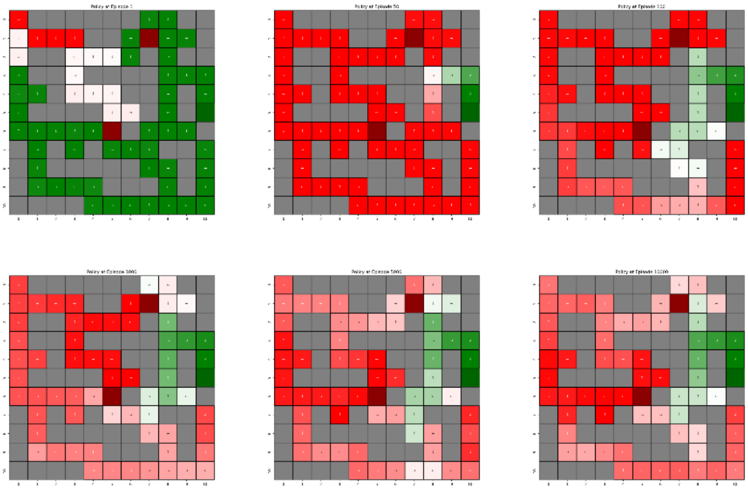

Figure 28. Policy plots for Q Learning when $\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$

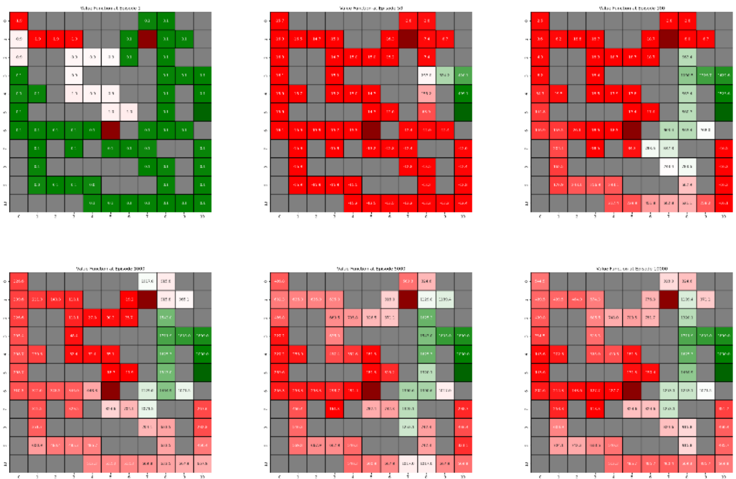

Figure 29. Value Function plots for Q Learning when $\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$

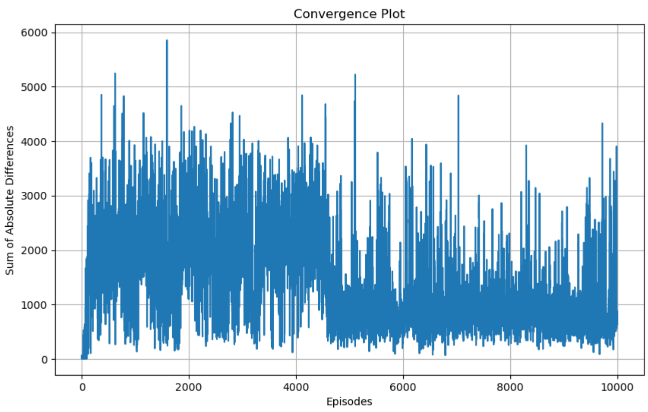

Figure 30. Convergence Plot for Q Learning when $\alpha = 1.0$, $\gamma = 0.95$, $\varepsilon = 0.2$

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$)

TD Learning

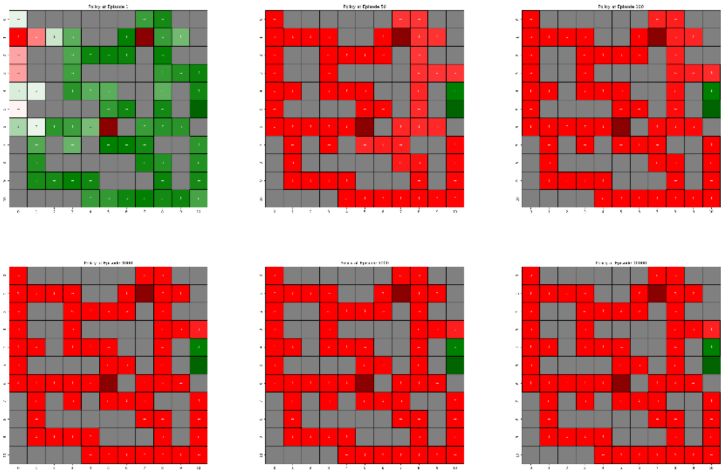

Figure 31. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$

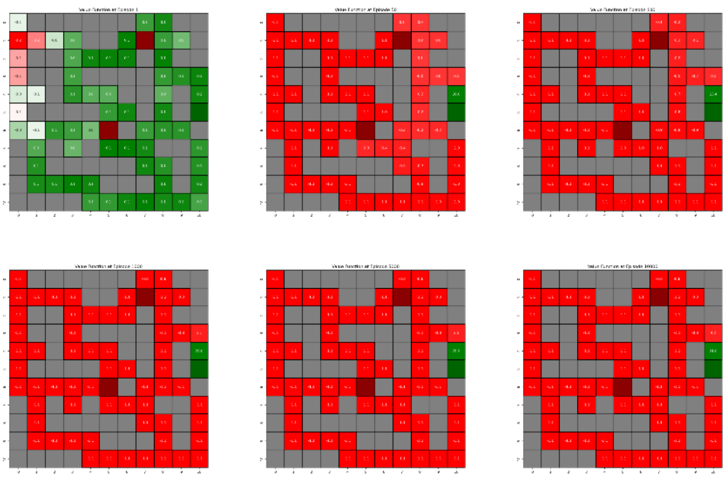

Figure 32. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$

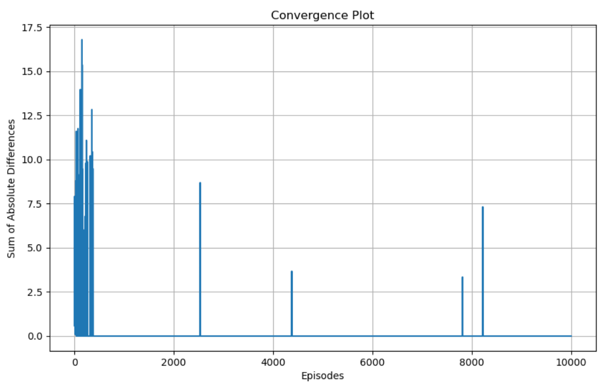

Figure 33. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$

Ezgi Sena Karabacak
2375178

26.05.2024

Q Learning

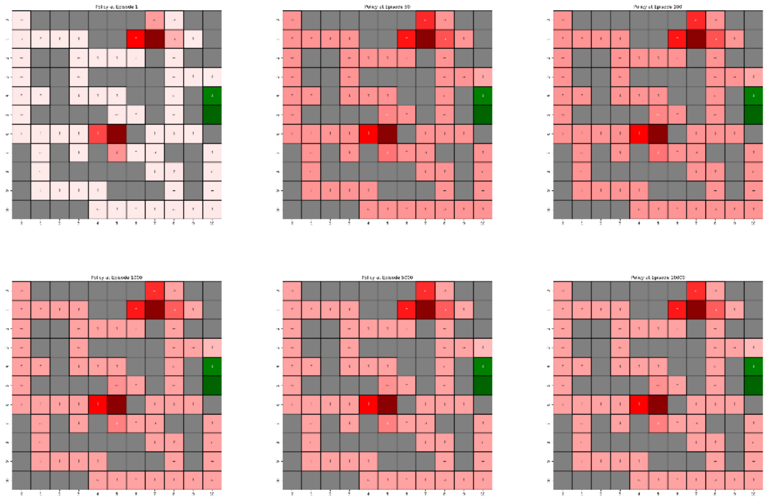

Figure 34. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$

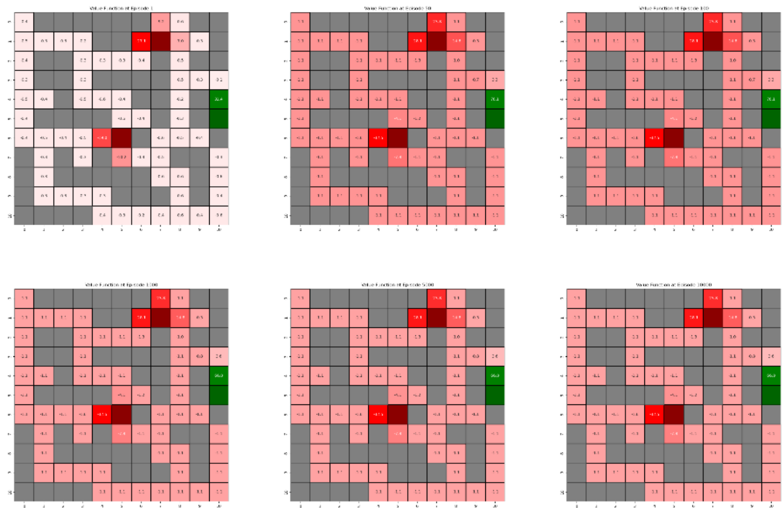

Figure 35. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$

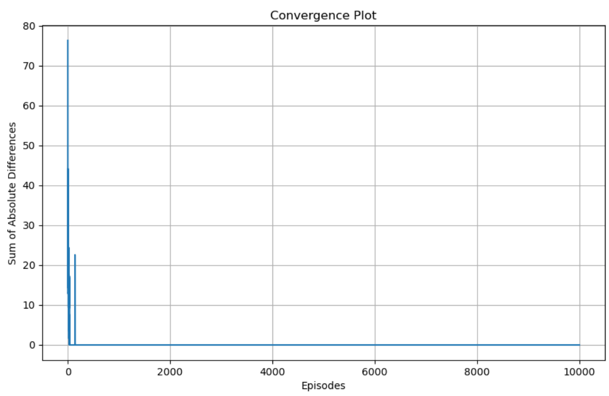

Figure 36. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.10$, $\varepsilon = 0.2$

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$)

TD Learning

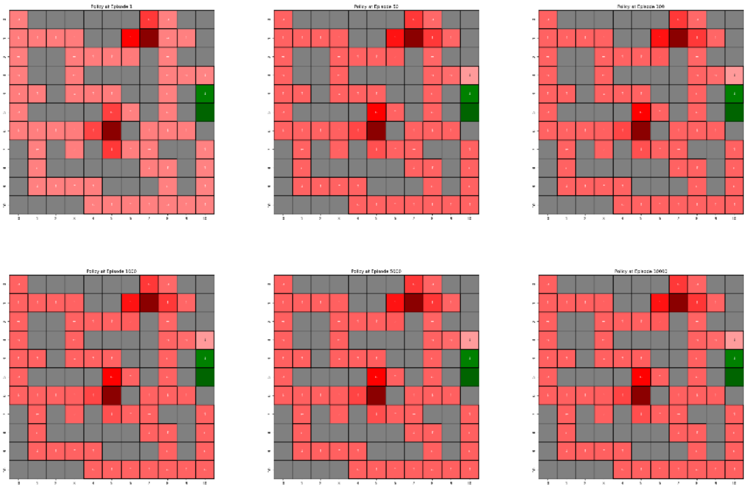

Figure 37. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$

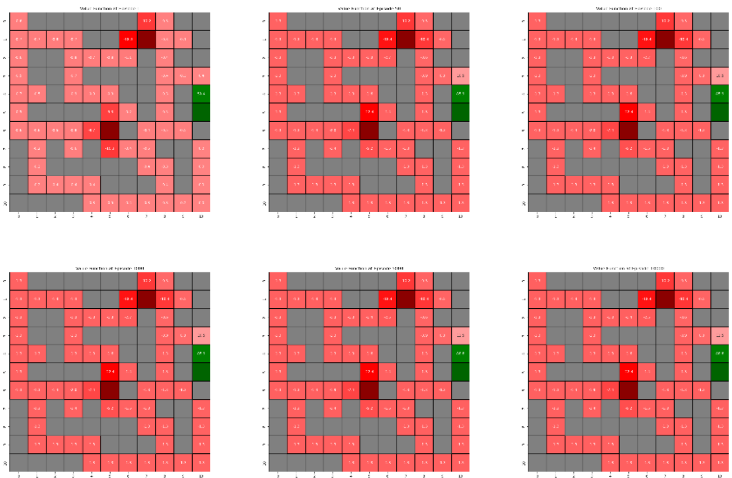

Figure 38. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$

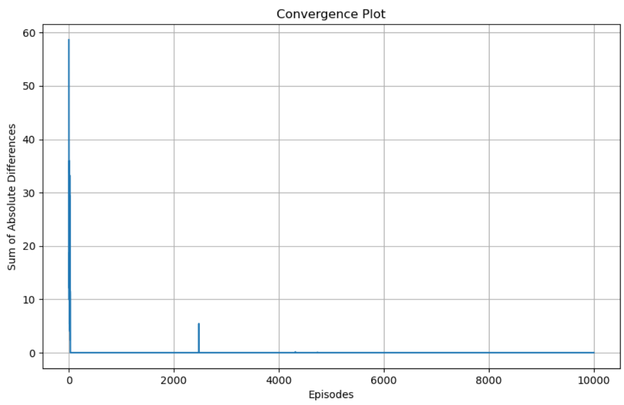

Figure 39. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$

Ezgi Sena Karabacak
2375178

26.05.2024

Q Learning

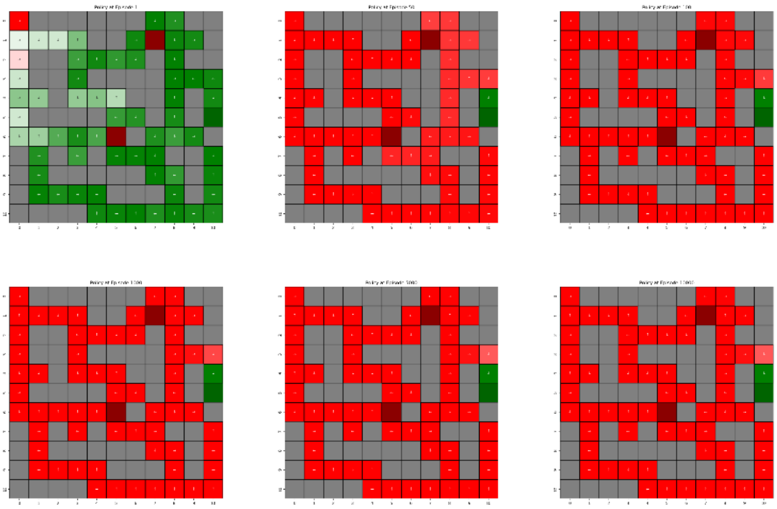

Figure 40. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$

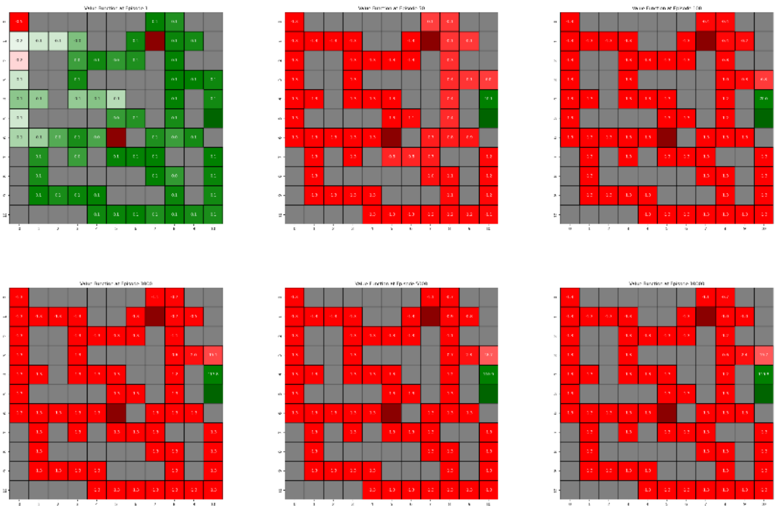

Figure 41. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$

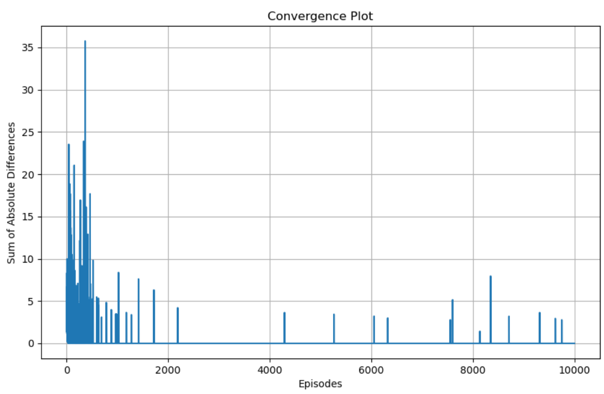

Figure 42. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.25$, $\varepsilon = 0.2$

Ezgi Sena Karabacak
2375178

26.05.2024

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.50$, $\varepsilon = 0.2$)

TD Learning

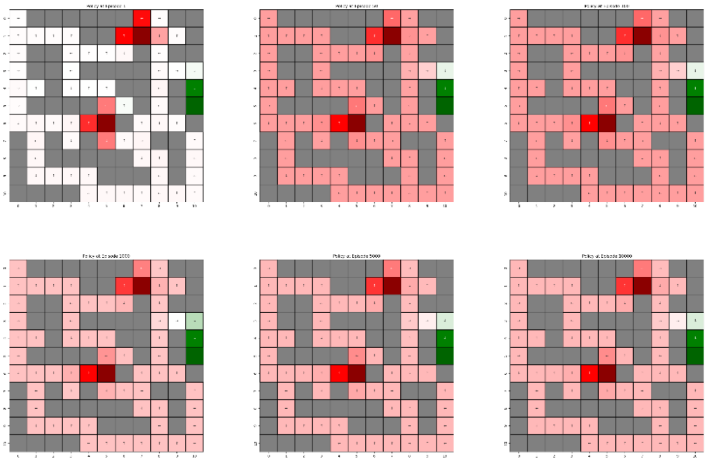

Figure 43. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.50$, $\varepsilon = 0.2$

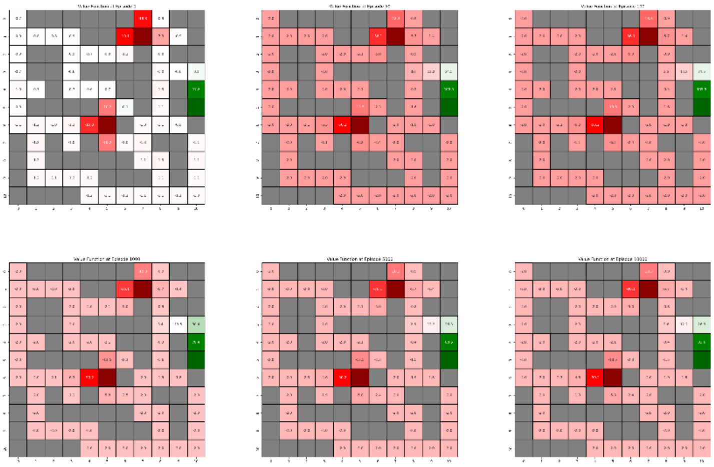

Figure 44. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.50$, $\varepsilon = 0.2$

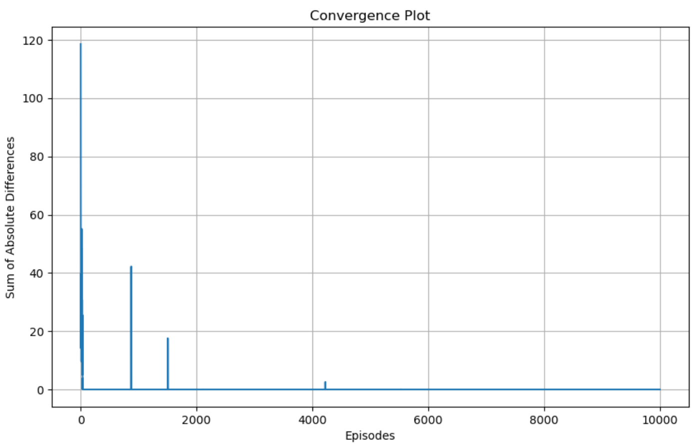

Figure 45. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.50$, $\varepsilon = 0.2$

Q Learning

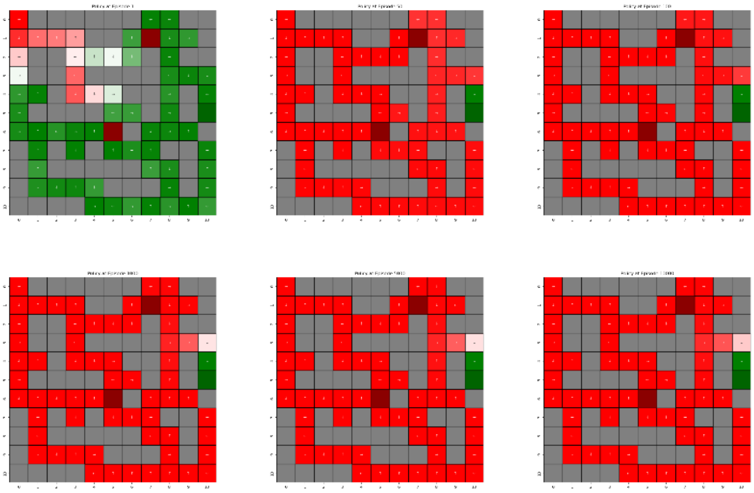

Figure 46. Policy plots for Q Learning when $\alpha = 0.1, \gamma = 0.50, \varepsilon = 0.2$

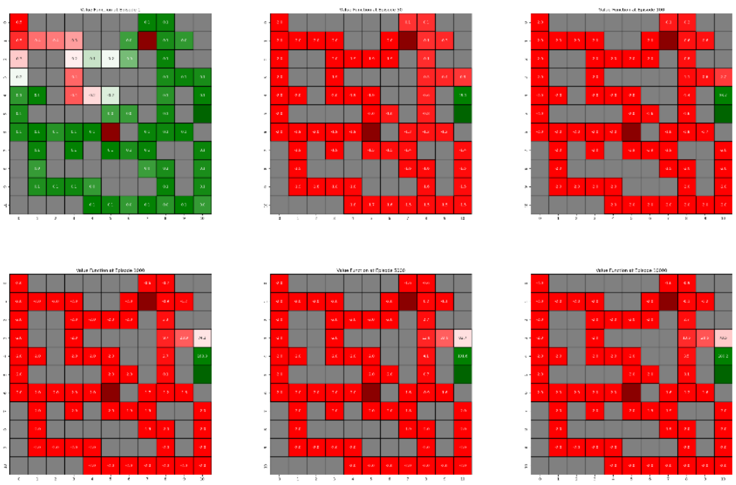

Figure 47. Value Function plots for Q Learning when $\alpha = 0.1, \gamma = 0.50, \varepsilon = 0.2$

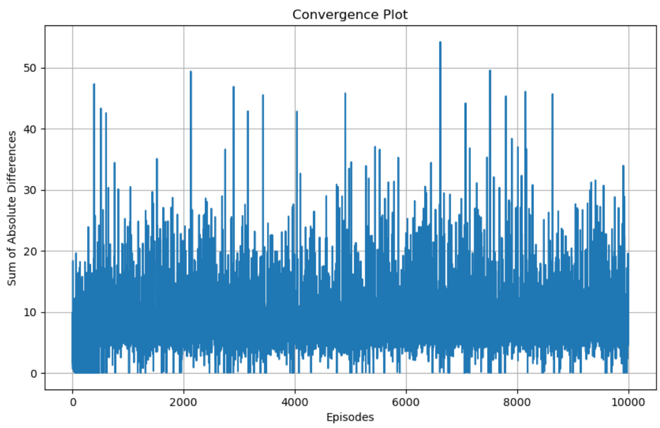

Figure 48. Convergence Plot for Q Learning when $\alpha = 0.1, \gamma = 0.50, \varepsilon = 0.2$

Ezgi Sena Karabacak
2375178 26.05.2024

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$)

TD Learning

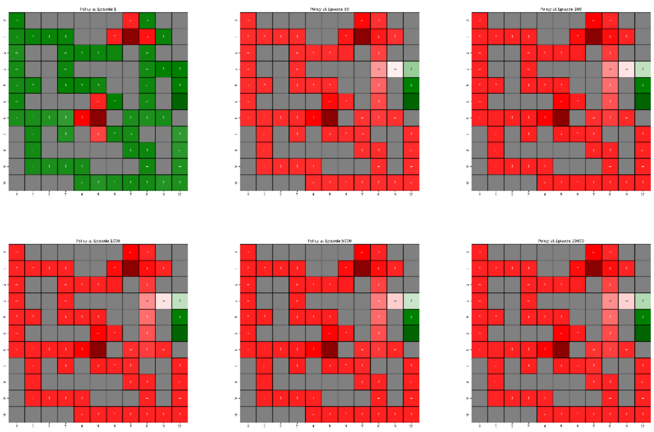

Figure 49. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$

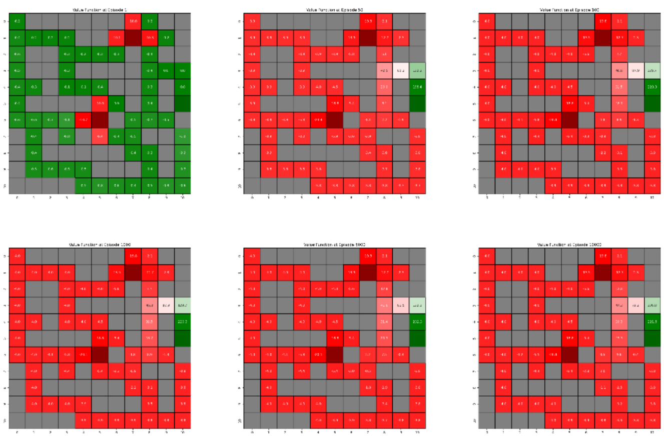

Figure 50. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$

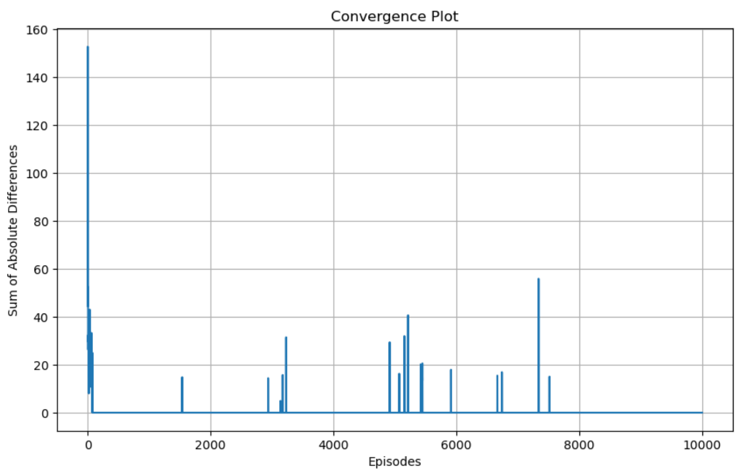

Figure 51. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$

Q Learning

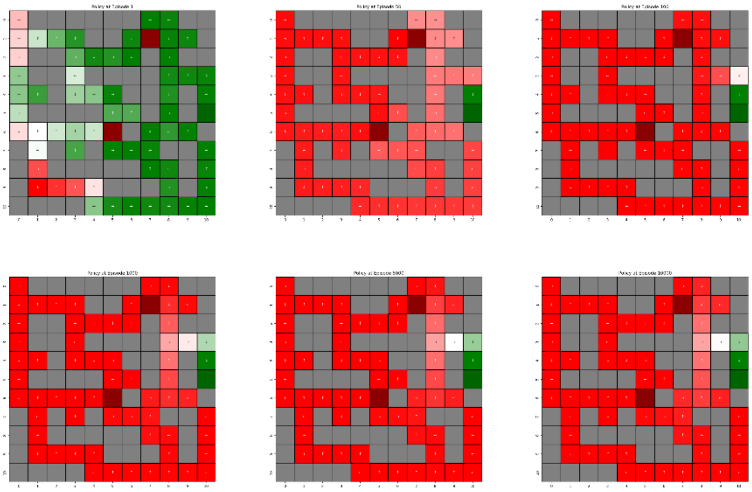

Figure 52. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$

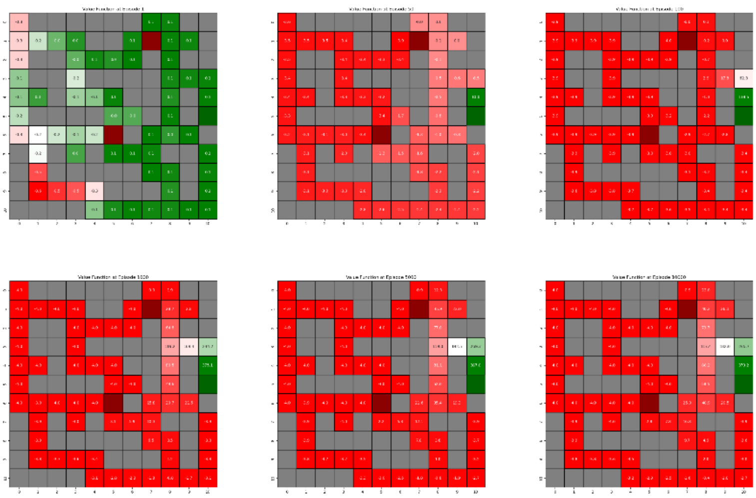

Figure 53. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$

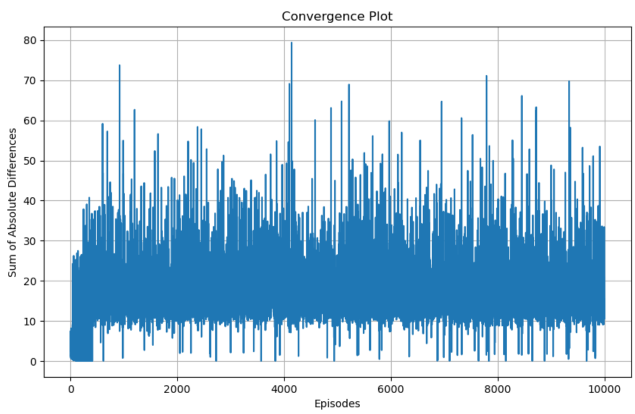

Figure 54. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.75$, $\varepsilon = 0.2$

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0$)

TD Learning

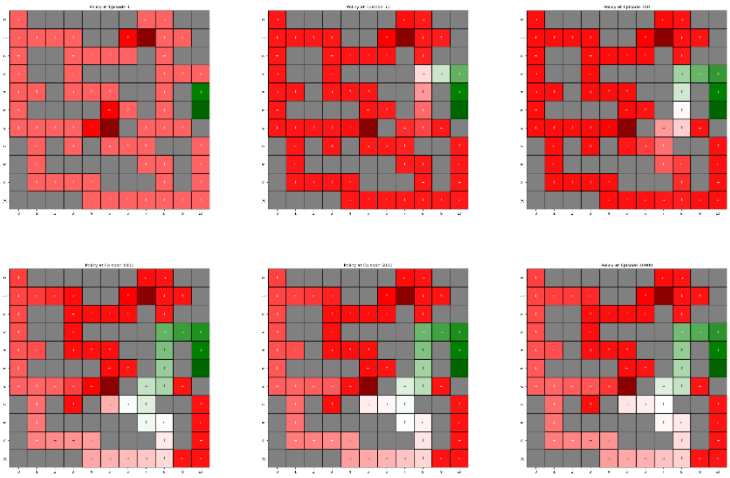

Figure 55. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0$

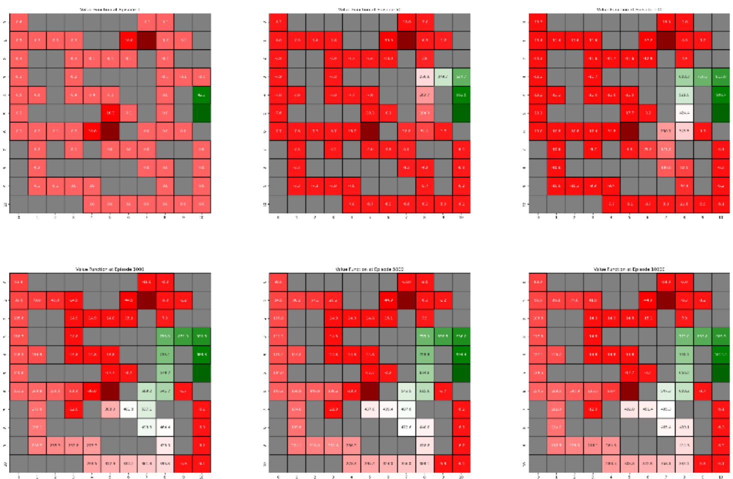

Figure 56. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0$

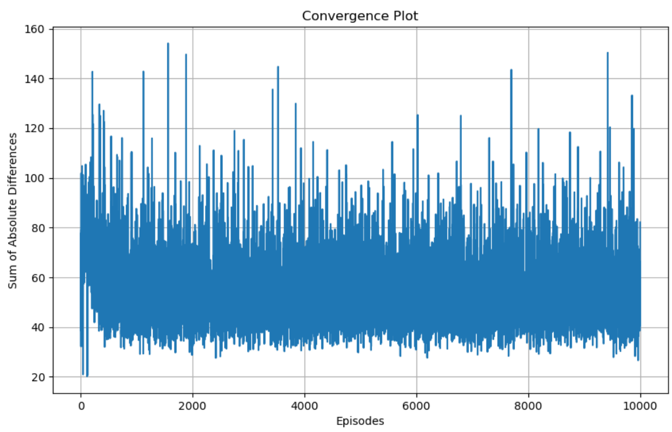

Figure 57. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0$

Ezgi Sena Karabacak
2375178

26.05.2024

Q Learning

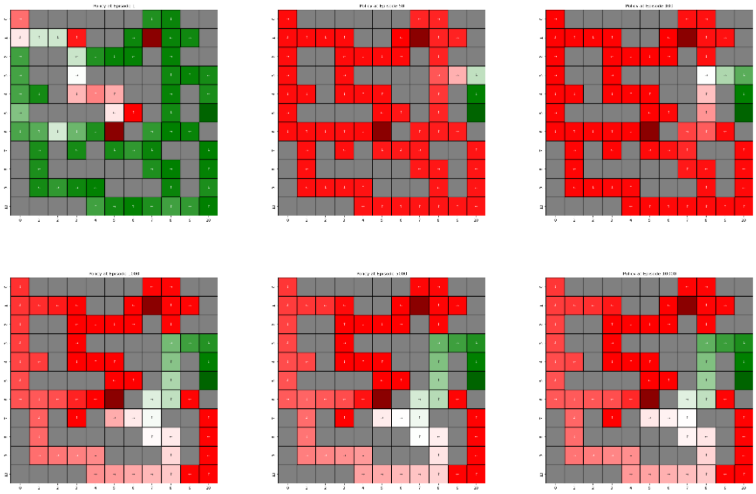

Figure 58. Policy plots for Q Learning when $\alpha = 0.1$ *,* $\gamma = 0.95$ *,* $\varepsilon = 0$

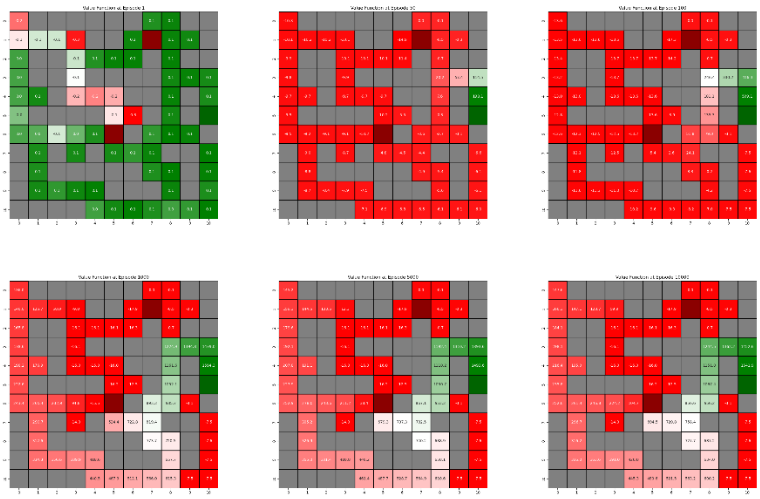

Figure 59. Value Function plots for Q Learning when $\alpha = 0.1$ *,* $\gamma = 0.95$ *,* $\varepsilon = 0$

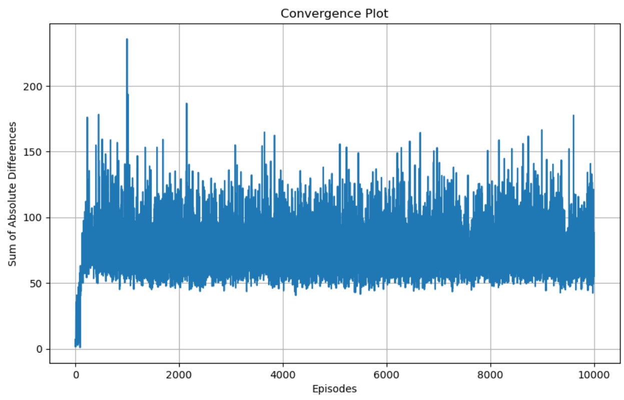

Figure 60. Convergence Plot for Q Learning when $\alpha = 0.1$ *,* $\gamma = 0.95$ *,* $\varepsilon = 0$

Ezgi Sena Karabacak 26.05.2024
2375178

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$)

TD Learning

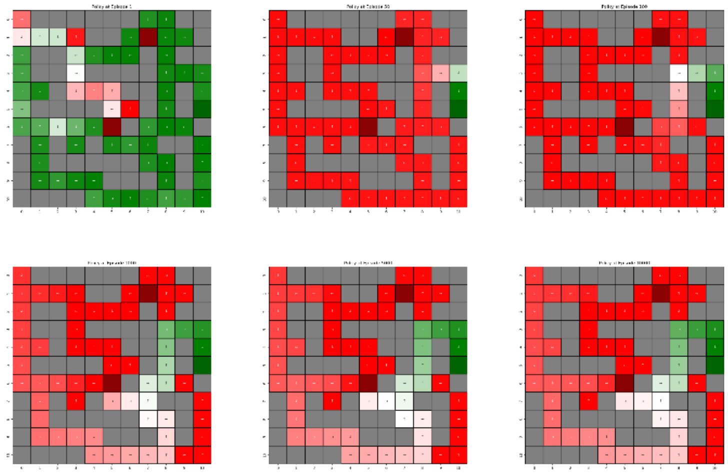

Figure 61. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$

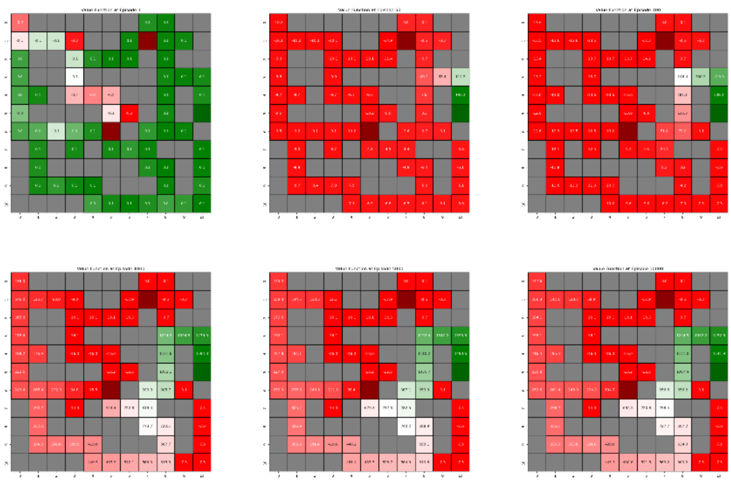

Figure 62. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$

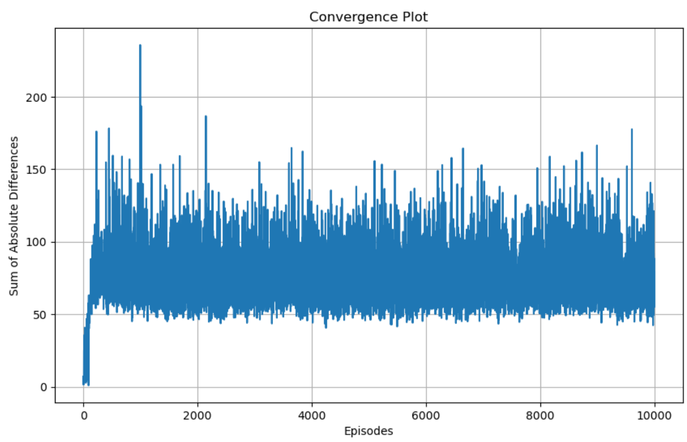

Figure 63. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$

Ezgi Sena Karabacak
2375178

26.05.2024

Q Learning

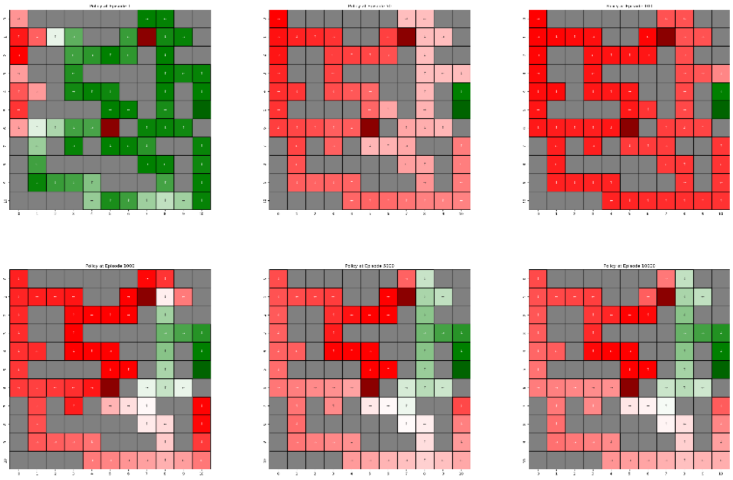

Figure 64. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$

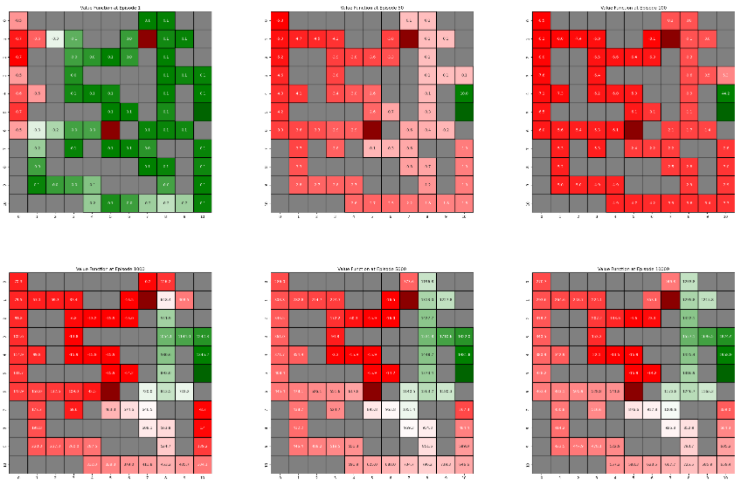

Figure 65. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$

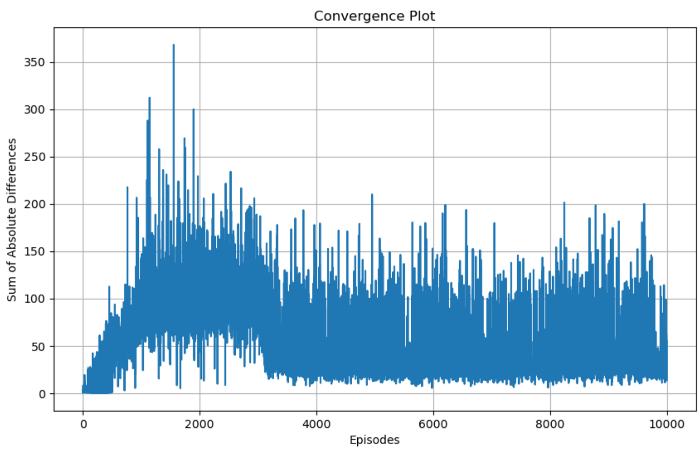

Figure 66. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.5$

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$)

TD Learning

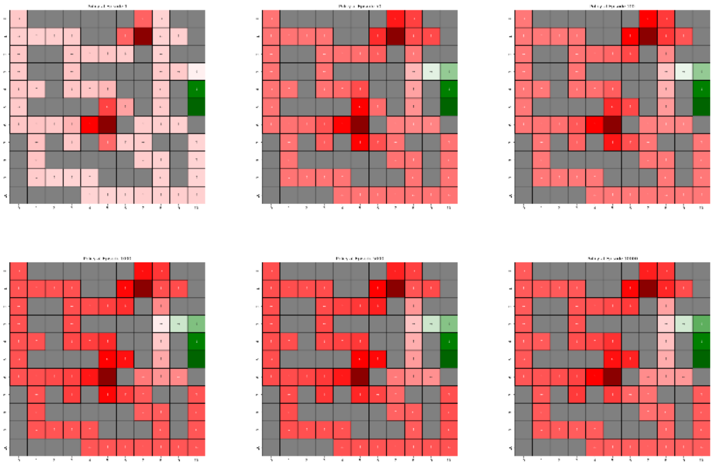

Figure 67. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$

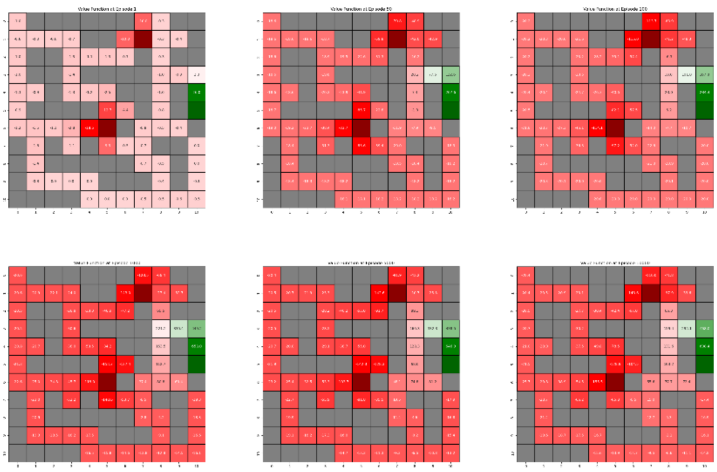

Figure 68. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$

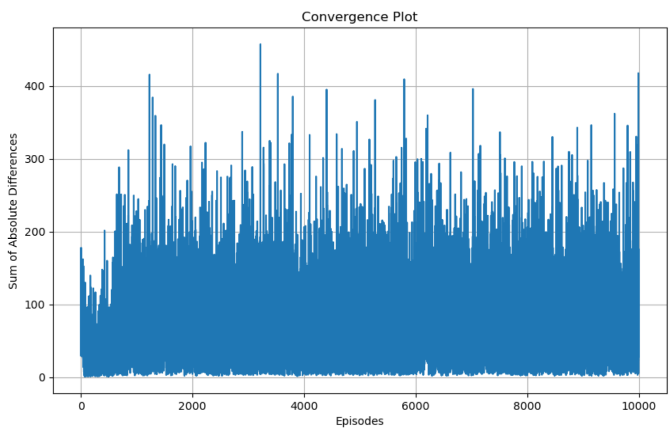

Figure 69. Convergence Plot for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$

Q Learning

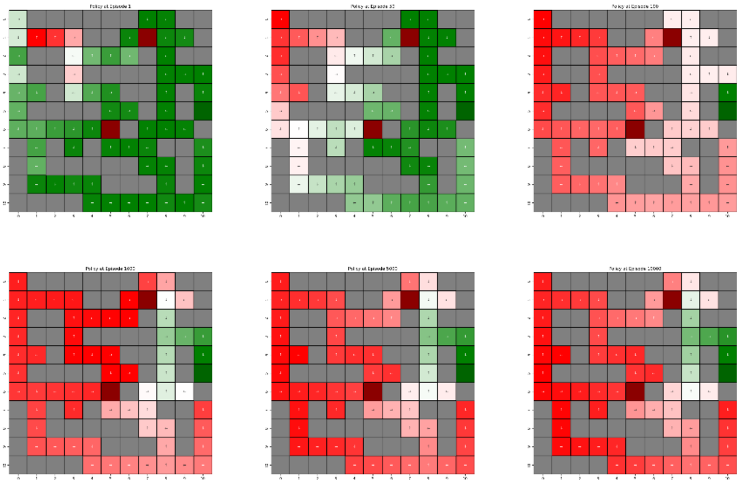

Figure 70. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$

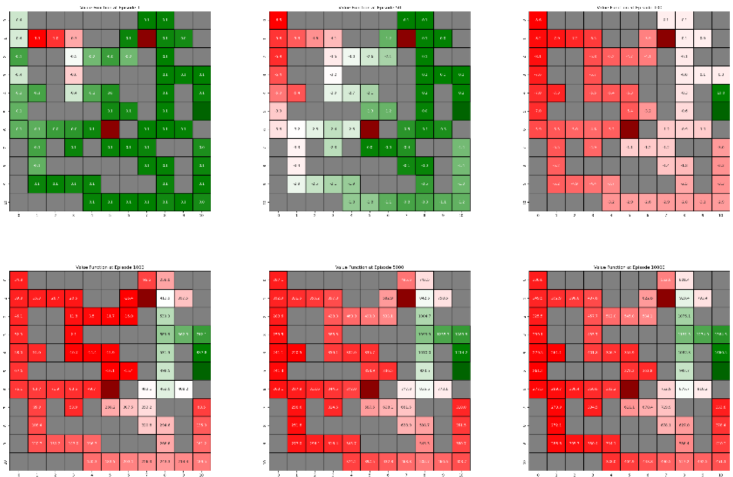

Figure 71. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$

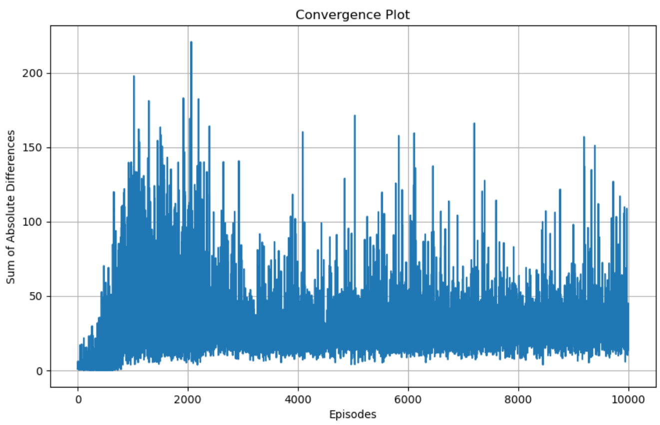

Figure 72. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 0.8$

Hyperparameter ($\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$)

TD Learning

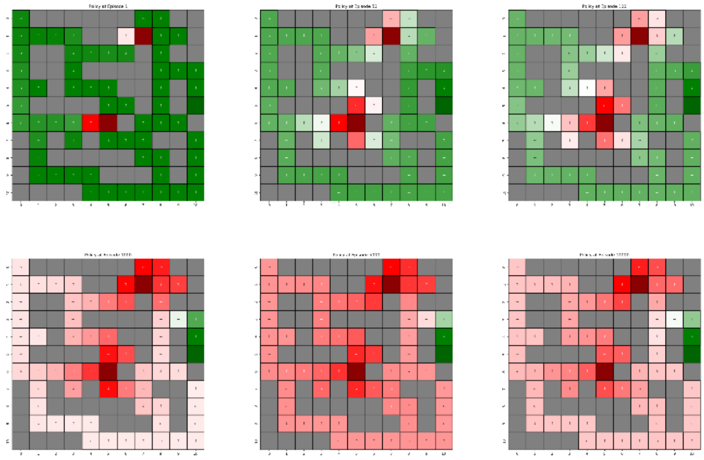

Figure 73. Policy plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$

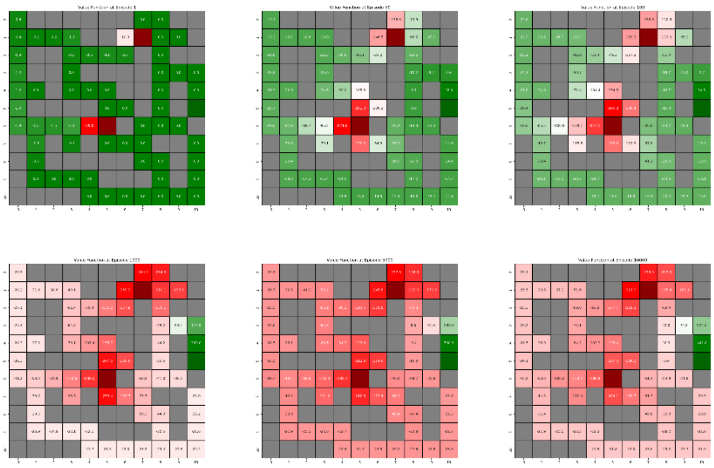

Figure 74. Value Function plots for TD Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$

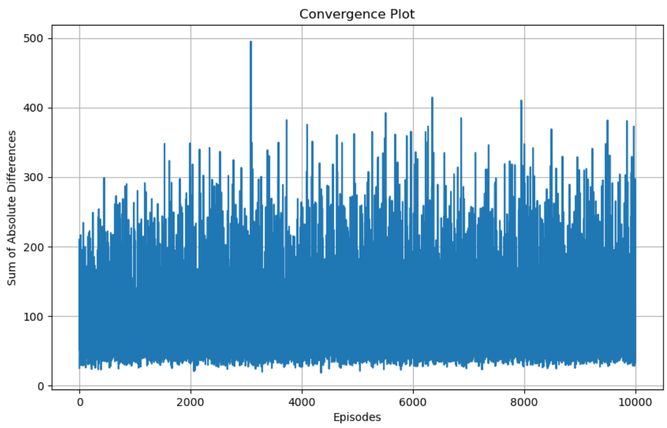

Figure 75. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$

Ezgi Sena Karabacak
2375178

26.05.2024

Q Learning

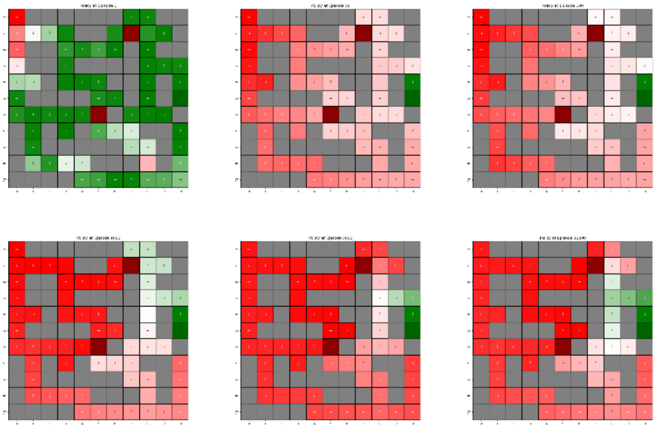

Figure 76. Policy plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$

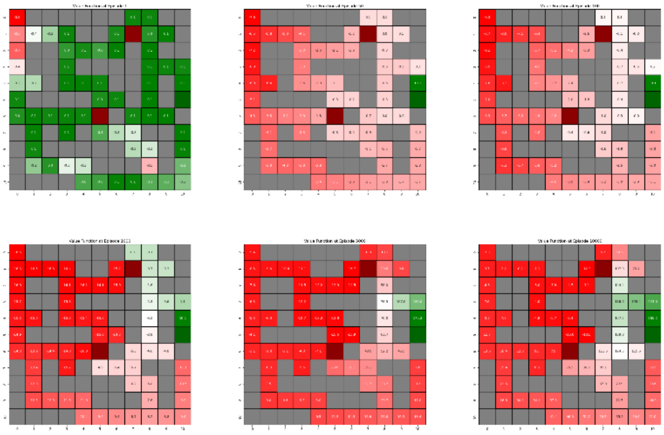

Figure 77. Value Function plots for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$

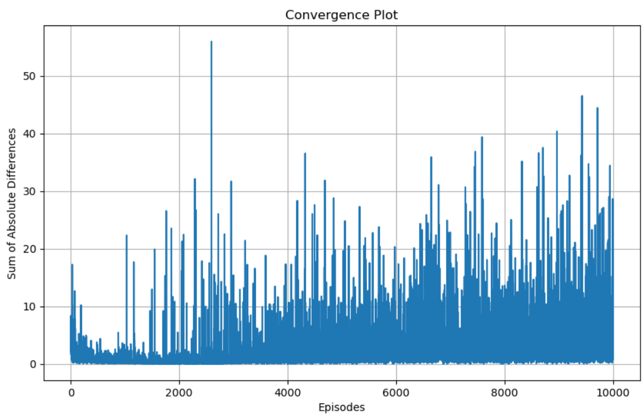

Figure 78. Convergence Plot for Q Learning when $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon = 1.0$

Ezgi Sena Karabacak
2375178 26.05.2024

3. Discussions

1. The transition probabilities (75% for the chosen direction, 5% for the opposite, 10% for perpendicular) introduce stochasticity, ensuring exploration while favoring intended actions. The reward function (state penalty of -1, trap penalty of -100, goal reward of 100) drives the agent towards the goal while avoiding traps, balancing exploration and exploitation effectively.

2. Initially, the utility values are uniform. Over time, values near the goal increase and those near traps decrease, showing the agent learning the maze's structure. By later episodes, the utility values stabilize, indicating that the agent has learned the optimal paths.

3. Yes, the utility value function converged, typically around 5000 episodes, as evidenced by the stabilization of utility values and consistent policy plots.

4. The process is sensitive to both parameters. High learning rates (e.g., 0.5, 1.0) cause instability, while low rates (e.g., 0.001, 0.01) slow convergence. Lower discount factors (e.g., 0.1, 0.25) lead to short-sighted strategies, whereas higher factors (e.g., 0.95) promote long-term planning.

5. Challenges included state representation, parameter tuning, and balancing exploration and exploitation. Solutions involved efficient state indexing, systematic experiments for optimal parameters, and implementing an ε-decay strategy.

6. Q Learning stabilized faster than TD(0) Learning because it directly estimates the action-value function, providing more granular information for policy improvement.

7. High ε (e.g., 1.0) promoted exploration but slowed convergence. Low ε (e.g., 0.0) led to insufficient exploration. Moderate ε (e.g., 0.2) balanced exploration and exploitation, improving performance.

8. TD(0) Learning is simpler and suitable for smaller mazes due to lower computational demands. Q Learning is more efficient for larger, complex mazes, providing faster convergence and better handling of large state spaces.

9. Increasing maze size, adding dynamic obstacles, and introducing multiple goals and traps can improve learning and add complexity, requiring the agent to develop more sophisticated strategies.

10. Using neural networks for function approximation, experience replay, adaptive exploration strategies, and multi-agent collaboration can enhance performance and reliability, making the algorithms more robust and efficient.

4. Code

```
import argparse
import os
import shutil
import numpy as np
from utils import plot_value_function, plot_policy, plot_convergence,
combine_plots

class MazeEnvironment:
    def __init__(self):
        self.maze = np.array([
            [0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1],
            [0, 0, 0, 0, 1, 1, 0, 2, 0, 0, 1],
            [0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1],
            [0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0],
            [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0],
            [0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 3],
            [0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 1],
            [1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0],
            [1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0],
            [1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0],
            [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
        ])
        self.start_pos = (0, 0)  # Start position of the agent
        self.current_pos = self.start_pos
        self.state_penalty = -1
        self.trap_penalty = -100
        self.goal_reward = 100
        self.actions = {0: (-1, 0), 1: (1, 0), 2: (0, -1), 3: (0, 1)}

    def reset(self):
        self.current_pos = self.start_pos
        return self.current_pos

    def step(self, action):
        row, col = self.current_pos
        action_prob = np.random.rand()

        if action_prob <= 0.75:
            next_pos = (row + self.actions[action][0], col +
self.actions[action][1])
        elif action_prob <= 0.80:
            next_pos = (row - self.actions[action][0], col -
self.actions[action][1])
        else:
            next_pos = (row + self.actions[(action + 2) % 4][0], col +
self.actions[(action + 2) % 4][1])
```

```
        if (0 <= next_pos[0] < self.maze.shape[0]) and (0 <= next_pos[1] <
self.maze.shape[1]) and self.maze[next_pos] != 1:
            self.current_pos = next_pos
        else:
            next_pos = self.current_pos

        reward = self.state_penalty
        if self.maze[next_pos] == 2:
            reward = self.trap_penalty
        elif self.maze[next_pos] == 3:
            reward = self.goal_reward

        done = self.maze[next_pos] in (2, 3)

        return next_pos, reward, done

class MazeTD0(MazeEnvironment):
    def __init__(self, maze, alpha=0.1, gamma=0.95, epsilon=0.2,
episodes=10000):
        super().__init__()
        self.maze = maze
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.episodes = episodes
        self.utility = np.zeros(self.maze.shape)

    def choose_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(list(self.actions.keys()))
        else:
            action_values = []
            for action in self.actions:
                next_pos = (state[0] + self.actions[action][0], state[1] +
self.actions[action][1])
                if (0 <= next_pos[0] < self.maze.shape[0]) and (0 <=
next_pos[1] < self.maze.shape[1]) and self.maze[next_pos] != 1:
                    action_values.append(self.utility[next_pos])
                else:
                    action_values.append(float('-inf'))
            return np.argmax(action_values)

    def update_utility_value(self, current_state, reward, new_state):
        current_value = self.utility[current_state]
        new_value = reward + self.gamma * self.utility[new_state]
        self.utility[current_state] += self.alpha * (new_value -
current_value)
```

Ezgi Sena Karabacak
2375178


```
    def run_episodes(self):
        value_history = []
        for episode in range(self.episodes+1):
            state = self.reset()
            done = False
            steps = 0
            while not done and steps < 1000:
                action = self.choose_action(state)
                new_state, reward, done = self.step(action)
                self.update_utility_value(state, reward, new_state)
                state = new_state
                steps += 1
                if done:
                    # Perform one last update for the terminal state
                    action = self.choose_action(state)
                    new_state, reward, done = self.step(action)
                    self.update_utility_value(state, reward, new_state)
                    state = new_state
                    steps += 1

            value_history.append((episode, self.utility.copy()))
            if episode % 1000 == 0:
                print(f"Episode {episode} completed")
        return value_history

class MazeQLearning(MazeEnvironment):
    def __init__(self, maze, alpha=0.1, gamma=0.95, epsilon=0.2,
episodes=10000):
        super().__init__()
        self.maze = maze
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.episodes = episodes
        self.q_table = np.random.rand(*self.maze.shape, len(self.actions)) *
0.1

    def choose_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(list(self.actions.keys()))
        else:
            state_actions = self.q_table[state[0], state[1], :]
            return np.argmax(state_actions)

    def update_q_table(self, action, current_state, reward, new_state):
        current_q = self.q_table[current_state[0], current_state[1], action]
        max_future_q = np.max(self.q_table[new_state[0], new_state[1], :])
```

```
        new_q = current_q + self.alpha * (reward + self.gamma * max_future_q -
current_q)
        self.q_table[current_state[0], current_state[1], action] = new_q

    def run_episodes(self):
        value_history = []
        for episode in range(self.episodes+1):
            state = self.reset()
            done = False
            steps = 0
            while not done and steps < 1000:
                action = self.choose_action(state)
                new_state, reward, done = self.step(action)
                self.update_q_table(action, state, reward, new_state)
                state = new_state
                steps += 1
                if done:
                    # Perform one last update for the terminal state
                    action = self.choose_action(state)
                    new_state, reward, done = self.step(action)
                    self.update_q_table(action, state, reward, new_state)
                    state = new_state
                    steps += 1
                    break
            utility_values = np.max(self.q_table, axis=2)
            value_history.append((episode, utility_values.copy()))
            if episode % 1000 == 0:
                print(f"Episode {episode} completed")
        return value_history

# Running experiments for TD(0) and Q-Learning
def run_experiments(args):
    maze = MazeEnvironment()
    alpha = args.alpha
    gamma = args.gamma
    epsilon = args.epsilon
    episodes = args.episodes

    # TD(0) Learning Experiments
    folder_name = f"TD_Learning_alpha_{alpha}_gamma_{gamma}_epsilon_{epsilon}"
    experiment_folder = os.path.join("results", folder_name)
    plot_folder = os.path.join(experiment_folder, "plots")

    # Remove the experiment folder if it exists
    if os.path.exists(experiment_folder):
        shutil.rmtree(experiment_folder)
    os.makedirs(plot_folder, exist_ok=True)
```

```
    print(f"TD(0) Learning with alpha={alpha}, gamma={gamma}, 
epsilon={epsilon}")
    maze_td0 = MazeTD0(maze.maze, alpha=alpha, gamma=gamma, epsilon=epsilon, 
episodes=episodes)
    td0_value_history = maze_td0.run_episodes()
    for episode, utility_values in td0_value_history:
        if episode in [1, 50, 100, 1000, 5000, 10000]:
            plot_value_function(utility_values, maze.maze, episode, 
plot_folder)
            plot_policy(utility_values, maze.maze, episode, plot_folder)
    plot_convergence(td0_value_history, plot_folder)
    combine_plots(plot_folder)

    # Q-Learning Experiments
    folder_name = f"Q_Learning_alpha_{alpha}_gamma_{gamma}_epsilon_{epsilon}"
    experiment_folder = os.path.join("results", folder_name)
    plot_folder = os.path.join(experiment_folder, "plots")

    # Remove the experiment folder if it exists
    if os.path.exists(experiment_folder):
        shutil.rmtree(experiment_folder)
    os.makedirs(plot_folder, exist_ok=True)

    print(f"Q-Learning with alpha={alpha}, gamma={gamma}, epsilon={epsilon}")
    maze_q_learning = MazeQLearning(maze.maze, alpha=alpha, gamma=gamma, 
epsilon=epsilon, episodes=episodes)
    q_learning_value_history = maze_q_learning.run_episodes()
    for episode, utility_values in q_learning_value_history:
        if episode in [1, 50, 100, 1000, 5000, 10000]:
            plot_value_function(utility_values, maze.maze, episode, 
plot_folder)
            plot_policy(utility_values, maze.maze, episode, plot_folder)
    plot_convergence(q_learning_value_history, plot_folder)
    combine_plots(plot_folder)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run TD(0) and Q-Learning 
experiments on a maze.")
    parser.add_argument('--alpha', type=float, default=0.1, help="Learning 
rate (default: 0.1)")
    parser.add_argument('--gamma', type=float, default=0.95, help="Discount 
factor (default: 0.95)")
    parser.add_argument('--epsilon', type=float, default=0.2, 
help="Exploration rate (default: 0.2)")
    parser.add_argument('--episodes', type=int, default=10000, help="Number of 
episodes (default: 10000)")
    args = parser.parse_args()
```

```
    run_experiments(args)
```

Adjusted Utils.py for Plotting

```
import os
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
from PIL import Image

def plot_value_function(value_function, maze, episode, folder_path):
    mask = np.zeros_like(value_function, dtype=bool)
    mask[maze == 1] = True  # Mask obstacles
    mask[maze == 2] = True  # Mask the trap
    mask[maze == 3] = True  # Mask the goal

    trap_position = tuple(np.array(np.where(maze == 2)).transpose(1, 0))
    goal_position = np.where(maze == 3)
    obs_position = tuple(np.array(np.where(maze == 1)).transpose(1, 0))

    plt.figure(figsize=(10, 10))
    cmap = LinearSegmentedColormap.from_list('rg', ["r", "w", "g"], N=256)
    ax = sns.heatmap(value_function, mask=mask, annot=True, fmt=".1f",
cmap=cmap,
                     cbar=False, linewidths=1, linecolor='black')
    ax.add_patch(plt.Rectangle(goal_position[::-1], 1, 1, fill=True,
edgecolor='black', facecolor='darkgreen'))
    for t in trap_position:
        ax.add_patch(plt.Rectangle(t[::-1], 1, 1, fill=True,
edgecolor='black', facecolor='darkred'))
    for o in obs_position:
        ax.add_patch(plt.Rectangle(o[::-1], 1, 1, fill=True,
edgecolor='black', facecolor='gray'))
    ax.set_title(f"Value Function at Episode {episode}")

    # Save the figure
    os.makedirs(folder_path, exist_ok=True)
    plt.savefig(os.path.join(folder_path,
f"value_function_episode_{episode}.png"))
    plt.close()

def plot_policy(value_function, maze, episode, folder_path):
    policy_arrows = {'up': '↑', 'down': '↓', 'left': '←', 'right': '→'}
    policy_grid = np.full(maze.shape, '', dtype='<U2')
    actions = ['up', 'down', 'left', 'right']
```

```
    trap_position = tuple(np.array(np.where(maze == 2)).transpose(1, 0))
    goal_position = np.where(maze == 3)
    obs_position = tuple(np.array(np.where(maze == 1)).transpose(1, 0))

    for i in range(maze.shape[0]):
        for j in range(maze.shape[1]):
            if maze[i][j] == 1 or (i, j) == goal_position:
                continue  # Skip obstacles and the goal
            best_action = None
            best_value = float('-inf')
            for action in actions:
                next_i, next_j = i, j
                if action == 'up':
                    next_i -= 1
                elif action == 'down':
                    next_i += 1
                elif action == 'left':
                    next_j -= 1
                elif action == 'right':
                    next_j += 1
                if 0 <= next_i < maze.shape[0] and 0 <= next_j <
maze.shape[1]:
                    if value_function[next_i][next_j] > best_value:
                        best_value = value_function[next_i][next_j]
                        best_action = action
            if best_action:
                policy_grid[i][j] = policy_arrows[best_action]

    mask = np.zeros_like(value_function, dtype=bool)
    mask[maze == 1] = True  # Mask obstacles
    mask[maze == 2] = True  # Mask the trap
    mask[maze == 3] = True  # Mask the goal

    plt.figure(figsize=(10, 10))
    cmap = LinearSegmentedColormap.from_list('rg', ["r", "w", "g"], N=256)
    ax = sns.heatmap(value_function, mask=mask, annot=policy_grid, fmt="",
cmap=cmap,
                     cbar=False, linewidths=1, linecolor='black')
    ax.add_patch(plt.Rectangle(goal_position[::-1], 1, 1, fill=True,
edgecolor='black', facecolor='darkgreen'))
    for t in trap_position:
        ax.add_patch(plt.Rectangle(t[::-1], 1, 1, fill=True,
edgecolor='black', facecolor='darkred'))
    for o in obs_position:
        ax.add_patch(plt.Rectangle(o[::-1], 1, 1, fill=True,
edgecolor='black', facecolor='gray'))
    ax.set_title(f"Policy at Episode {episode}")
```

```
    # Save the figure
    os.makedirs(folder_path, exist_ok=True)
    plt.savefig(os.path.join(folder_path, f"policy_episode_{episode}.png"))
    plt.close()

def plot_convergence(value_history, folder_path):
    episodes = [vh[0] for vh in value_history]
    diffs = [np.sum(np.abs(value_history[i+1][1] - value_history[i][1])) for i
in range(len(value_history)-1)]

    plt.figure(figsize=(10, 6))
    plt.plot(episodes[1:], diffs, label='Sum of Absolute Differences')

    plt.xlabel('Episodes')
    plt.ylabel('Sum of Absolute Differences')
    plt.title('Convergence Plot')
    plt.grid(True)

    # Save the figure
    os.makedirs(folder_path, exist_ok=True)
    plt.savefig(os.path.join(folder_path, "convergence.png"))
    plt.close()

def combine_plots(folder_path):
    episodes = [1, 50, 100, 1000, 5000, 10000]
    plot_types = ["policy", "value_function"]

    for plot_type in plot_types:
        images = []
        output_path = os.path.join(folder_path,
f"combined_{plot_type}_plots.png")

        for episode in episodes:
            plot_path = os.path.join(folder_path,
f"{plot_type}_episode_{episode}.png")

            if os.path.exists(plot_path):
                image = Image.open(plot_path)
                images.append(image)
            else:
                print(f"Missing {plot_type} plot for episode {episode}")

        # Create a new image with a suitable size
        if images:
            width, height = images[0].size
            combined_image = Image.new('RGB', (width * 3, height * 2))  # 3
columns, 2 rows
```

```
        for i, image in enumerate(images):
            row = i // 3
            col = i % 3
            combined_image.paste(image, (col * width, row * height))

        combined_image.save(output_path)
    else:
        print(f"No {plot_type} images to combine.")
```