Reinforcement Learning: Assignment 2

Sameer Ahamed (202381922)

July 19, 2024

1 Problem 1

1.1 Problem Summary

The problem entails an agent navigating a grid that is subject to specific rules for transitions and rewards. The specifics are as follows:

1.1.1 Gridworld Layout

- **Grid**: 5x5 cells.
- **Agent Actions**: Move to the left, right, up, or down. The agent remains in the same state if it tries to exit the grid.
- **Rewards possible**:
  - **Blue Square (0,1)**: Any action results is a reward of 5 and transfers the agent to the red square (3,2).
  - **Green Square (0,4)**: Any action results is a reward of 2.5 and advances the agent to either the yellow square (4,4) or the red square (3,2) with 0.5 probability each.
  - **Red Square (3,2)**: No transitions or special rewards are present in the red square.
  - **Yellow Square (4,4)**: No transitions or special rewards are available in the yellow square.
  - **White Squares**: A reward of 0 is obtained by moving between white squares. The reward for exiting the grid is -0.5.

1.1.2 Objectives

- **Estimation of the Value Function:**
  - Utilize a reward discount factor ($\gamma = 0.95$) and a policy that allows the agent to travel in any direction with an equal probability (0.25).
  - Estimate the value function for each state by employing the following methods:
    - Explicitly resolving the system of Bellman equations.
    - Iterative policy evaluation.
    - Value iteration.
  - Determine the states with the highest values and evaluate whether the results are consistent with the anticipated outcomes.
- **Determine the Most Effective Policy:**
  - By explicitly resolving the Bellman optimality equation.
  - Using iterative policy evaluation in conjunction with policy iteration.
  - Implementing policy enhancement through value iteration.
- This problem design enables the application and comparison of various reinforcement learning algorithms to determine the optimal policy and value function for the gridworld environment.

1.2 Solving using system of Bellman Equations

1.2.1 Approach

To solve this problem, I used the system of Bellman equations explicitly. The steps involved are as follows:

1. **Define Constants**:
   - Discount factor ($\gamma = 0.95$).
   - Number of states (25).
   - Actions ('up', 'down', 'left', 'right') with equal probability (0.25 each).
2. **Initialize Transition and Reward Matrices**:
   - P (state transition probability matrix) and R (reward vector) initialized to zeros.
3. **Setup Special States**:
   - Define rewards and transitions for blue, green, red, and yellow squares.
4. **Populate Transition Probabilities and Rewards**:
   - For each state, calculate the possible next states and update the transition matrix P and reward vector R.
5. **Solve the Bellman Equation**:
   - Use the linear equation $V = R + \gamma PV$ to solve for the value function V.
6. **Reshape and Output the Value Function**:
   - Reshape the value function vector V to a 5x5 grid V_{grid}.

1.2.2 Value Function

The value function obtained is:

$$\begin{bmatrix} 3.20497185 & 5.00000000 & 2.58699068 & 1.82323264 & 2.50000000 \\ 2.08467462 & 2.29281086 & 1.48236902 & 0.76654570 & -0.01490769 \\ 1.19512002 & 1.08689682 & 0.59520652 & -0.06313312 & -1.20914409 \\ 0.66539284 & 0.49327025 & 0.00000000 & -0.41843179 & -1.69868494 \\ 0.44787096 & 0.32463769 & 0.10111666 & 0.00000000 & -1.72083366 \end{bmatrix}$$

- The states with the highest values are around the blue and green squares due to their high immediate rewards (5 and 2.5, respectively).
- The high value of the blue square (5) directly impacts the surrounding states, resulting in higher values as the agent can easily transition to the blue square and gain a high reward.
- The value of the green square also positively impacts its surrounding states, but to a lesser extent due to the lower reward (2.5) and the probabilistic nature of the transitions.

The results align with the expected outcomes, where states close to high-reward states have higher values.

1.2.3 Optimal Policy

The attached figure 1 indicates the best actions for the agent to maximize its rewards. The arrows in the image represent the optimal moves from each state. States around the blue and green squares show movements towards these high-reward states, demonstrating the effectiveness of the derived policy.

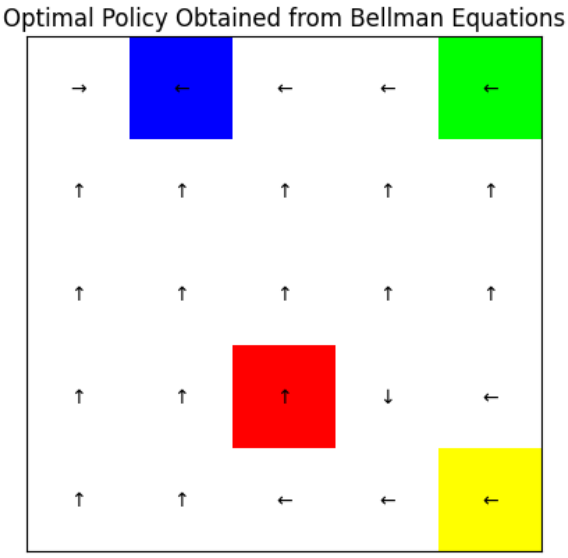

Figure 1: Policy diagram generated explicitly using Bellman Equations

1.3 Solving using Iterative Policy Evaluation

1.3.1 Approach

To solve this problem, I used iterative policy evaluation. The steps involved are as follows:

1. **Define Parameters**:
   - Discount factor ($\gamma = 0.95$).
   - Small threshold for convergence ($\theta = 1 \times 10^{-6}$).
   - Actions ('up', 'down', 'left', 'right') with equal probability (0.25 each).
2. **Initialize Reward and Transition Structures**:
   - Initialize rewards for each cell.
   - Define special states with their respective rewards and transitions.
3. **Initialize the Value Function**:
   - Set the initial value function V to zero for all states.
4. **Define Possible Movements**:
   - Specify how movements change the agent's position on the grid.
5. **Iterative Policy Evaluation**:
   - Repeatedly update the value function until it converges (change is less than θ).

1.3.2 Value Function

The value function obtained is:

$$\begin{bmatrix} 2.376320 & 5.000000 & 2.337893 & 1.606109 & 2.288008 \\ 1.305552 & 1.998917 & 1.426075 & 1.056872 & 0.956763 \\ 0.342589 & 0.684867 & 0.610845 & 0.461039 & 0.253148 \\ -0.364209 & -0.068702 & 0.000000 & 0.020350 & -0.078746 \\ -0.916873 & -0.609928 & -0.446298 & -0.296607 & 0.000000 \end{bmatrix}$$

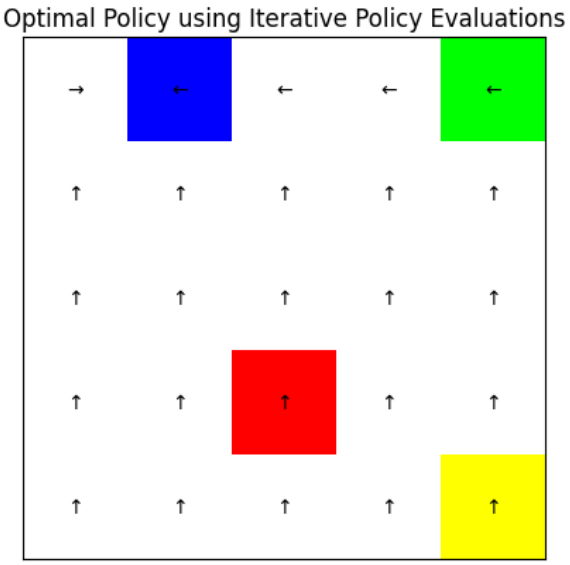

Figure 2: Policy diagram generated using Iterative Policy

- The states with the highest values are around the blue and green squares due to their high immediate rewards (5 and 2.5, respectively).
- The high value of the blue square (5) directly impacts the surrounding states, resulting in higher values as the agent can easily transition to the blue square and gain a high reward.
- The value of the green square also positively impacts its surrounding states, but to a lesser extent due to the lower reward (2.5) and the probabilistic nature of the transitions.

1.3.3 Optimal Policy

The agent's most effective course of action for optimizing its rewards is illustrated in the figure of the optimal policy. The optimal movements from each state are denoted by the arrows in the image. The derived policy's efficacy is demonstrated by the movement of states around the blue and green squares toward these high-reward states.

1.4 Solving using Value Iteration

1.4.1 Approach

To solve this problem, I used value iteration. The steps involved are as follows:

1. **Define Parameters**:
   - Discount factor ($\gamma = 0.95$).
   - Small threshold for convergence ($\theta = 1 \times 10^{-6}$).
   - Actions ('up', 'down', 'left', 'right') with equal probability (0.25 each).
2. **Initialize the Value Function**:
   - Set the initial value function V to zero for all states.
3. **Define Special States and Possible Movements**:
   - Specify rewards and transitions for special states (blue, green, red, yellow).
   - Define how movements change the agent's position on the grid.
4. **Value Iteration**:

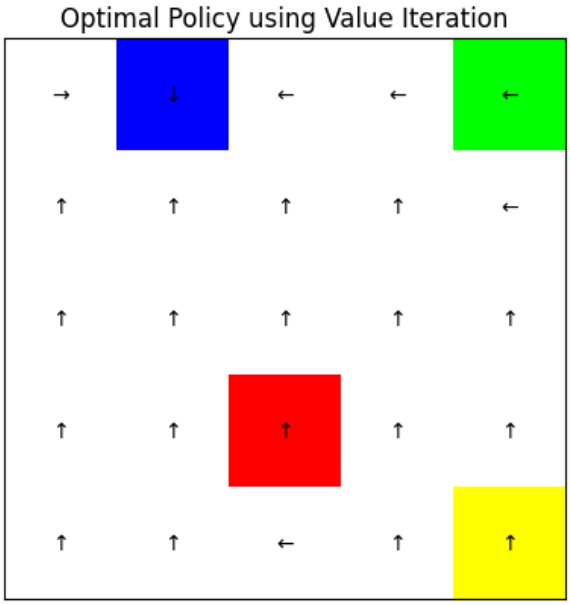

Figure 3: Policy diagram generated using Value Iteration

- Repeatedly update the value function by considering the maximum expected value of possible actions until convergence.

1.4.2 Value Function

The value function obtained is:

$$\begin{bmatrix} 4.750000 & 5.000000 & 4.750000 & 4.512500 & 2.500000 \\ 4.512500 & 4.750000 & 4.512500 & 4.286875 & 4.072531 \\ 4.286875 & 4.512500 & 4.286875 & 4.072531 & 3.868905 \\ 4.072531 & 4.286875 & 0.000000 & 3.868905 & 3.675459 \\ 3.868905 & 4.072531 & 3.868905 & 3.675459 & 0.000000 \end{bmatrix}$$

1.4.3 Optimal Policy

The figure 3 of the optimal policy indicates the best actions for the agent to maximize its rewards. The arrows in the image represent the optimal moves from each state. States around the blue and green squares show movements towards these high-reward states.

2 Problem 2

2.1 Problem Summary

The gridworld environment was modified by adding terminal states represented by black squares. The agent's movement in the grid terminates upon reaching a black square. Additionally, any move from a white square to another white square yields a reward of -0.2.

2.2 Solving using Monte Carlo with Exploring Starts

1. **Parameters**:
   - Discount factor ($\gamma = 0.95$).
   - Epsilon for epsilon-soft policy ($\epsilon = 0.1$).
   - Actions: 'up', 'down', 'left', 'right'.

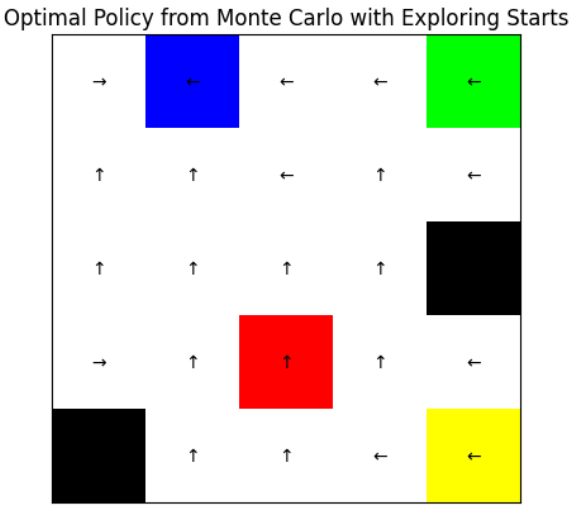

Figure 4: Optimal Policy derived from Monte Carlo with Exploring Starts

- Grid dimensions: 5x5.

2. **Initialization**:
   - Value function Q initialized to zero.
   - Returns sum and count matrices initialized to zero.
3. **Special and Terminal States**:
   - Blue square: (0,0) with a reward of 5, transitions to (3,2).
   - Green square: (0,4) with a reward of 2.5, transitions to (4,4) or (3,2).
   - Red square: (3,2) with a reward of 0.
   - Yellow square: (4,4) with a reward of 0.
   - Terminal states (black squares): (4,0) and (2,4).
4. **Movement and Reward**:
   - Defined possible movements and associated rewards for transitions.
5. **Policy**:
   - Epsilon-greedy policy with exploring starts, generating episodes, and updating Q values.

2.2.1 Optimal Policy

The arrows in figure 4 indicate the optimal actions from each state, considering the terminal states. We can infer that in every state, the agent is optimized to take the shortest path to the high reward states. This is a direct consequence of the penalty we imposed on the grid. And the states surrounding tiles near the black tiles are never such that it leads to the black tile as the reward is far less due to termination.

2.3 Solving using On-policy First-Visit MC Control for epsilon soft policy without Exploring Starts

2.3.1 Approach

Initialization

- Define parameters such as the discount factor (γ) and ϵ for the ϵ-soft policy.

- Define actions, grid dimensions, and initialize the Q-value function, return sums, return counts, and policy.
- Normalize initial policy probabilities to ensure they sum to 1.

Special and Terminal States

- Specify the special states (blue, green, red, and yellow squares) with their respective rewards and transitions.
- Define terminal states (black squares).

Movement and Rewards

- Define possible movements (up, down, left, right) and their effects on the agent's state and reward.
- Include penalties for moving off-grid and special handling for special and terminal states.

Episode Generation

- Randomly initialize state and generate episodes using the current policy until reaching a terminal state.
- Select actions based on the current policy's probabilities.

Policy Update

- Update the policy based on the observed returns to ensure it remains ϵ-soft.
- Normalize the policy probabilities after each update.

Monte Carlo Control Loop

- Run multiple episodes (I have chosen 5000).
- For each episode, calculate the return G and update the Q-values and policy for each state-action pair encountered.

Extract and Plot Policy

- Extract the optimal policy from the Q-values.
- Plot the optimal policy and value function for visualization.

Functions

get_next_state

Determines the next state and reward based on the current state and action.

generate_episode

Generates an episode by following the current policy.

update_policy

Updates the policy to be ϵ-soft based on the Q-values.

monte_carlo_on_policy

Runs the Monte Carlo control loop to update Q-values and policy.

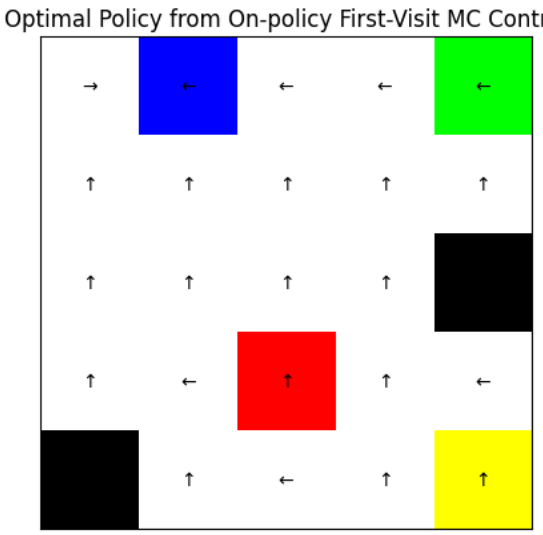

Figure 5: Policy Generated for Monte Carlo with epsilon-soft without exploring start

extract_policy

Extracts the optimal policy from the Q-values.

plot_policy

Plots the optimal policy and value function.

2.3.2 Optimal Policy

The On-policy First-Visit Monte Carlo (MC) control algorithm was successfully implemented and applied to the modified gridworld problem. The algorithm utilized an ϵ-soft policy to balance exploration and exploitation while updating the action-value function $Q(s, a)$ based on observed returns.

The generated optimal policy from figure 5 effectively guides the agent to maximize rewards, as demonstrated by the directional arrows in the output visualization. Special states (blue, green, red, yellow) and terminal states (black) were appropriately handled, leading to a coherent policy that converges towards optimal actions.

The value function grid provided insight into the expected returns for each state under the optimal policy. This solution highlights the effectiveness of the Monte Carlo control approach in solving reinforcement learning problems with stochastic transitions and rewards.

2.4 Off-Policy Prediction via Importance Sampling with equiprobable moves

Policies

- **Target Policy**: This is the optimal policy we aim to learn, assumed to be a uniform distribution over actions initially.
- **Behavior Policy**: This is an exploratory policy used to generate episodes. It is also a uniform distribution over actions.
- **Monte Carlo Sampling**: Episodes are generated using the behavior policy, recording states, actions, and rewards.
- **Returns Calculation**: For each episode, the returns are calculated by summing the rewards backward from the end of the episode.

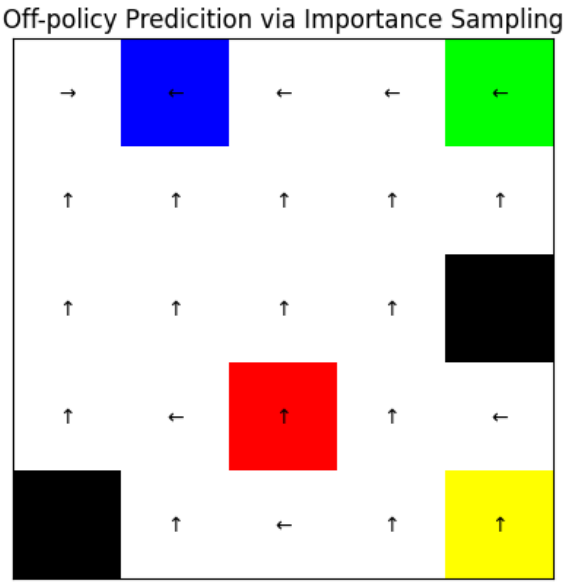

Figure 6: Off-policy Prediction via Importance Sampling using Equiprobable moves

- **Importance Sampling**: The importance sampling ratio is computed to adjust the returns from the behavior policy to the target policy. This ratio corrects the bias introduced by using a different policy for generating episodes.
- **Value Estimation**: The state values are updated using the weighted returns.

Policy Plotting

The `plot_policy` function visualizes the derived policy by indicating the best action to take from each state, excluding the black terminal states.

Code Output Analysis

The output of the code is visualized in the provided image 6, showing the optimal policy derived from the value estimates.

Policy Arrows

- Arrows indicate the best action to take from each state based on the highest value of neighboring states.
- The directions are consistent with moving towards states with higher rewards.

Special States

- **Blue State (0, 1)**: The policy directs the agent towards the blue state due to its high reward (5), followed by a jump to the red state.
- **Green State (0, 4)**: The policy also favors reaching the green state because of its positive reward (2.5), with a probability of jumping to either the yellow or red state.
- **Red State (3, 2)** and **Yellow State (4, 4)**: These states follow normal transitions.
- **Black States (4, 0) and (2, 4)**: The policy avoids directing actions towards these terminal states as they end the episode with no further rewards.

General Policy Trends

- The policy generally points away from the terminal states, indicating the agent's tendency to maximize rewards by moving towards states with higher expected returns.
- The arrows reflect the optimal paths to collect rewards efficiently and avoid negative rewards from unnecessary movements.

Rare Abnormalities Observed

- Environment gives penalty for taking longer path. Yet, it is found that for a few coordinates, a longer path is chosen to reach high reward states
- For example, on the coordinate (4,2), the most suitable direction is up based on visual observation. Yet, the policy generated shows the agent to take a left in that state.

2.5 Permutation of locations of green and blue squares

2.5.1 Policy Iteration Method

To determine a suitable policy, we use the policy iteration method, which consists of two main steps: policy evaluation and policy improvement.

2.5.2 Policy Evaluation

Policy evaluation involves calculating the value function $v_\pi(s)$ for a given policy π. This value function estimates the expected return starting from state s and following policy π. The Bellman expectation equation for the value function is used iteratively to update the value of each state until convergence:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{1}$$

where:

- $\pi(a|s)$ is the probability of taking action a in state s under policy π.
- $p(s', r|s, a)$ is the probability of transitioning to state s' and receiving reward r given action a in state s.
- γ is the discount factor.

2.5.3 Policy Improvement

Policy improvement involves improving the current policy π by making it greedy with respect to the value function v_π. This means selecting actions that maximize the expected return:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2}$$

This new policy π' is guaranteed to be at least as good as the old policy π. The process of policy evaluation and policy improvement is repeated until the policy converges to the optimal policy π^*.

2.5.4 Handling the Permutation of Special States

In this environment, the locations of the blue and green squares permute with a probability of 0.1. This dynamic element requires special consideration in the policy iteration process.

2.5.5 Modified Transition Probabilities

To handle permutations, adjust the transition probabilities to account for the dynamic positions of the blue and green squares. During value function evaluation, incorporate the probability of these permutations.

2.5.6 Stochastic Policy

Develop a stochastic policy that considers the expected value of actions given the probability of permutations. The policy should balance immediate rewards with long-term expected returns.

2.5.7 Comparison with Static Case

When the squares do not permute, the environment becomes deterministic. In this case, the policy iteration process follows a straightforward application of policy evaluation and improvement. The agent can reliably predict the outcomes of its actions based on fixed rewards and transitions.

2.5.8 Dynamic Permutation Case

- **Uncertainty in State Transitions**: The dynamic permutation introduces uncertainty in state transitions. The agent must account for the probability of the special states changing positions, leading to a more complex calculation of the expected value of actions.
- **Stochastic Decision-Making**: The policy must be more robust, potentially involving stochastic decision-making to handle the variations in the environment. The agent's strategy should consider the expected outcomes over multiple possible configurations of the grid.
- **Policy Flexibility**: The resulting policy in the dynamic case is likely to be more flexible, as it needs to adapt to the changing positions of the blue and green squares. This could result in a more generalized strategy that aims to maximize long-term rewards under uncertainty.

2.5.9 Comparison

In the static case where the blue and green squares do not permute:

- **Deterministic Transitions**: The transitions between states are deterministic, making it easier for the agent to predict the outcomes of its actions.
- **Simpler Policy**: The policy can be more straightforward and deterministic since the agent does not need to account for variations in the environment.
- **Stable Value Function**: The value function is more stable and converges faster because the state transitions are fixed.

In contrast, the dynamic case with permuting squares requires:

- **Handling Stochasticity**: The policy must handle the stochastic nature of state transitions due to the permutations.
- **Complex Policy Evaluation**: The value function must incorporate the expected rewards over multiple potential configurations of the grid.
- **Adaptive Strategy**: The policy must be adaptive, considering both immediate rewards and long-term expectations under uncertainty.

The dynamic nature of the gridworld environment requires incorporating the probability of permutations into the policy iteration process. This leads to a more complex but adaptive policy compared to the static case. By adjusting transition probabilities and developing a stochastic policy, we can effectively handle the dynamic permutations and achieve a robust strategy for the agent.