Memorial University of  
Newfoundland

A red logo with white text

Description automatically generated

Assignment 3

Experimentation with simple bandit learning algorithms

Submitted by:

Oviemuno Peter Utomakili (201477924)

Dhiraj Rajkarnikar (202382307)

Table of Contents

[INTRODUCTION 3](#_Toc174206022)

[OBJECTIVES 3](#_Toc174206023)

[EXPERIMENTS 4](#_Toc174206024)

[CONCLUSION 11](#_Toc174206025)

Table of Figures

[Figure 1: Agent's Movement in the Grid with 500 Episodes using Sarsa 6](#_Toc174206010)

[Figure 2: Agent's Movement in the Grid with 500 Episodes using Q-Learning 6](#_Toc174206011)

[Figure 3: Sum of Rewards over 500 Episodes 7](#_Toc174206012)

[Figure 4: Optimal Policy for Sarsa 8](#_Toc174206013)

[Figure 5: Optimal Policy for Q-Learning 8](#_Toc174206014)

[Figure 6: Value Function of Gradient Monte Carlo, Semi-Gradient TD (0) and Exact Value Function. 10](#_Toc174206015)

# INTRODUCTION

Reinforcement Learning is a very important area in artificial intelligence which primarily focuses on how agents can maximize some cumulative reward by taking some actions in an environment. Temporal Difference (TD) learning algorithms such as Sarsa and Q-learning are particularly significant due to their ability to learn from incomplete information about the environment. We take a grid world problem which is one of the most commonly used environments in RL research. In the grid world we assume we have a finite, two-dimensional grid for an agent to navigate from a starting point to one of the terminal states avoiding specific penalty states.

**Sarsa**, one of the popular reinforcement learning algorithms, is used to learn a policy that maximizes the expected cumulative reward for an agent interacting with an environment. It is an on-policy learning algorithm, meaning it learns the value of the policy it is currently following and maintains a Q-function Q (s, a) that estimates the expected cumulative reward and often uses epsilon-greedy policy to balance exploration and exploitation.

**Q-learning** is also another popular algorithm in reinforcement learning but is an off-policy Temporal Difference (TD) learning algorithm, meaning it learns the value of the optimal policy independently of the agent’s actions, even while the agent might be following a different policy. This also uses an epsilon-greedy policy for action selection to balance exploration and exploitation.

**Monte Carlo Method** is perhaps the most popular algorithm in reinforcement learning which estimates the value of a state by averaging the returns from multiple episodes that start form the state. This method relies on complete episodes to calculate the value function and does not require a model of the environment.

**Semi-Gradient TD (0)** is a temporal difference learning approach that updates the value function using the difference between the current state’s estimated value and the value of the next state, weighted by a linear function approximation. This method combines the aspects of Monte Carlo and dynamic programming.

# OBJECTIVES

The primary objectives of this project are to utilize the Sarsa and Q-learning algorithms to learn optimal policy that guides the agent from the start to the terminal state while minimizing penalties. We also aim to analyze the trajectories generated by the learned policies and compare the performance of these two algorithms in terms of sum of rewards over episodes. Since we are using epsilon-greedy action selection, we examine how this influences the agent’s exploration and exploitation during learning. In summary, we demonstrate how different reinforcement learning approaches – on-policy (Sarsa) and off-policy (Q-learning) handles the exploration-exploitation trade-off and learn optimal policies in a stochastic environment.

# EXPERIMENTS

**Scenario 1: Consider a grid world problem where the agent starts at the blue square and moves to a neighbouring state with equal probability. If the agent moves to a red state, it receives a reward of −20 and goes back to the start, i.e., the blue square. A move between any two other states receives a reward of −1. A move that attempts to move outside of the grid receives a reward of −1. The black squares serve as a terminal state. Intuitively, you can see how the goal here is to pass through the opening in the red “wall” and get to one of the black squares and hence terminate the episode.**

A grid of squares with red blue and white rectangles

Description automatically generated

**Use the Sarsa and Q-learning algorithms to learn the optimal policy for this task. Plot a trajectory of an agent utilizing the policy learned by each of the methods. Are they different or similar? Why or why not? You may assume to use ϵ-greedy action selection for this task. How does the sum of rewards over an episode behaves for each of these two methods.**

**ANSWER:**

1. **Environment Setup**

A 5\*5 grid world environment is set up with a starting state, terminal states and “red” states or penalty states. The agent starts at the bottom-left corner and tries to reach the black terminal states situated at the top-left and top-right corners avoiding the red or penalty states which penalizes the agent with -20 reward and sends it back to the starting state.

To implement this, we define a class – “grid world” with the possible actions are moving 'up', 'down', 'left', or 'right' with ‘reset’ method resetting the agent and step method defining the movement logic, updates the agent’s sate and returns the next state, reward and informs whether the episode is done.

1. **Q-learning**

The next step was to implement Q-learning algorithm with a function which initializes a Q-table to store the action-value function for each state-action pair. The algorithm runs for a specified number of episodes and for each episode, the agent starts from initial state, chooses an action using epsilon-greedy strategy and moves to the next state, receives a reward and updates the Q-value. This process is repeated until the agent reaches terminal state. This function returns the learned Q-values and cumulative rewards over episodes.

1. **Sarsa**

We then implement Sarsa algorithm which is similar to Q-learning algorithm but different in the sense that it is an on-policy algorithm learning algorithm so the Q-values are updated using actual actions taken in the next state and not the optimal one. This function also returns the learned Q-values and cumulative rewards over episodes.

1. **Learning**

An instance of the “grid world” class is created and both Q-learning and Sarsa algorithms are used to learn the polices according to environment.

1. **Results Visualization**

The experiment conducted provides us the learned Q-values and we utilize these Q-values to simulate the agent’s movement through the grid which are shown in the figure below.

|  |  |
| --- | --- |
| A black and red line on a white grid  Description automatically generated  Figure 1: Agent's Movement in the Grid with 500 Episodes using Sarsa | A black square with red dots and a red line  Description automatically generated  Figure 2: Agent's Movement in the Grid with 500 Episodes using Q-Learning |

In the figures above, the red lines with dots represent the path taken by the agent. Using Sarsa, the agent reaches the terminal state on the top-right whereas using Q-learning, the agent reaches the terminal state on the top-left. Both the algorithm successfully avoids any red penalty states. The agent explores different paths dur tot the epsilon-greedy policy which introduces randomness in action selection.

We also calculate the sum of rewards over the episodes for each of the algorithms which is shown in the figure below.

A graph with orange lines

Description automatically generated

Figure 3: Sum of Rewards over 500 Episodes

From the figure above, we can see that Sarsa has more significant and frequent dips compared to Q-learning specially around 300 to 400 episodes. This difference can be attributed to the on-policy nature of Sarsa. Since Sarsa learns the value of the policy it is following (which includes exploration actions), it may reinforce paths that, while safe, might also occasionally lead to larger penalties if the agent encounters unexpected situations. While Q-learning updates its Q-values on the maximum future rewards, regardless of the agent’s actual actions. This tends to favor more optimal paths, which is why its reward curve is smoother with fewer significant drops compared to Sarsa. Therefore, we can say that Q-learning appears to have slightly more stable learning curve suggesting that it might have converged to a more optimal policy likely due to its off-policy nature. Both ultimately both algorithms learn effective polices and reach the terminal states without encountering the penalty states.

The following two figures show the optimal policy learned by the two different algorithms for our grid world environment. E ach cell in the grid represents a state, and the arrows within each cell indicate the action that the optimal policy prescribes for that state. The arrows point in the direction that the agent should move to maximize its cumulative reward, based on the learned Q-values.

|  |  |
| --- | --- |
| Figure 4: Optimal Policy for Sarsa | A grid of black arrows  Description automatically generated  Figure 5: Optimal Policy for Q-Learning |

In the figures above, T is the terminal states, R represents the red or penalty states and arrows indicate the direction the agent should move when in that particular state to follow optimal policy.

1. **Conclusion (Scenario 1)**

From the first experiment, we can draw several conclusions about the performance of Sarsa and Q-learning in the environment we have set up. Sarsa learns the policy that the agent is actually following making it more conservative. This leads to it having safer policies like in our case where exploration might lead to significant penalties. Whereas the Q-learning focuses on learning the optimal policy regardless of the actions the agent actually takes during exploration as it updates the Q-values based on the maximum future rewards. This leads to it discovering more optimal paths and perform better in an environment where exploration can uncover significantly better outcomes.

The sum of rewards over the episodes showed that Q-learning had fewer and less severe negative spikes compared to Sarsa suggesting it was more effective in avoiding penalties and consistently finding paths that lead to higher cumulative rewards.

**Scenario 2: Consider a scenario where we have a random walk on a 7 × 7 grid. That is, we are equally likely to move up, down, left, or right. Suppose that we start the random walk at the precise center of the grid. We assume that the lower left and upper right corners are terminal states, with, respectively, rewards of −1 and 1. Rewards for transitions between two states are 0, if an attempt to transition outside the wall is made, the agent stays in the same spot and receives a reward of 0. Compute the value function for this “random walk” policy using (1) gradient Monte Carlo method and (2) the semi-gradient TD(0) method with an affine function approximation. How does it compare to the exact value function?**

**ANSWER:**

1. **Environment Setup**

A 7\*7 grid world environment is set up with the starting points at the center of the grid for random walk. The terminal states are in the lower-left corner and the upper-right corner of the grid with the reward of -1 and +1 respectively. The transition rewards for movement between any two non-terminal states gives reward of 0. This environment also adds a boundary condition which defines that if the agent attempts to move outside the grid, it stays in the same spot with a reward of 0. The agent moves randomly with equal probability in each of the four possible directions – up, down, left and right.

For this we set up a class – “Random Walk Grid” with initial state at center (3, 3) position of the grid, terminal rewards at (6, 0) with reward of -1 and (0, 60 with reward of +1. We also implement reset() method which resets the agent to starting state and step() function which randomly selects an action, computes next state and checks if it is within the grid boundaries and ends the episodes if it is a terminal state.

1. **Gradient Monte Carlo**

We then implement the Gradient Monte Carlo Method which is used to estimate the value function for each state using the Monte Carlo method with gradient descent. In this, for each episode, the agent starts at the initial state, and a trajectory of states and rewards is collected with agent’s random movement until it reaches the terminal state. The function returns the learned parameters of the value function which are used to estimate the value for each state in the grid.

1. **Semi-gradient TD (0)**

The next algorithm we implement is semi-gradient temporal difference method where we estimate the value function for each state using the TD (0) method with linear function approximation. For this, in each episode, the agent moves randomly until it reaches the terminal state and during these movements, at each step, the parameters are updated using the TD (0) update rule. The update is semi-gradient as the gradient is taken only with respect to the parameters of the current state’s value. This function return the learned parameters of the value function.

1. **Exact value function**

Then we implement the exact value function for comparison with the previous two methods. This function computes the exact value function for each state using an iterative approach. We update the value of each state based on expected value of the states it can transition to, averaging over the possible next states and this process is repeated until the values converge. This function returns the exact value function for all states in the grid.

1. **Learning**

An instance of the “Random Walk Grid” class is created and all the methods mentioned above are used to learn the value functions. The total episode is 5000 with learning rate of 0.01 for Gradient Monte Carlo and Semi-Gradient TD (0) but the later with additional discount factor of 1.0.

1. **Results Visualization**

The results as the value function for all three methods is shown in the figure below. The heatmap shows the values in the grid representing expected cumulative reward from each state assuming the agent follows the policy.

**A chart of different colors

Description automatically generated with medium confidence**

Figure 6: Value Function of Gradient Monte Carlo, Semi-Gradient TD (0) and Exact Value Function.

From the figure we can see that the values of the heatmap from the Gradient Monte Carlo is increasing smoothly with the highest value at upper-right corner. The value decreases as the agent moves towards the lower-left corner indicated by the darkest blue color. The values appear relatively smooth across the grid as it averages over many episodes to estimate the value function.

For the Semi-Gradient TD (0) value function, the heat map also shows a gradient of values increasing towards the upper-right corner. This method produces highest values overall compared to Gradient Monte Carlo as the values in this matrix are more pronounced might be the result of bootstrapping nature of the TD (0) algorithm. The overall pattern is similar to Monte Carlo method, but TD (0) shows more significant difference between states closer to the terminal states and those farther away.

The exact value function shows the sharpest transitions compared to the other two. This method identifies the terminal states, and the values are exactly 0 for most of the grid, except near terminal states. This reflects that under the random walk policy, states far from terminal states have an expected value close to 0. Overall, this method provides a clear and accurate representation of the expected returns under random walk policy.

1. **Conclusion (Scenario 2)**

From our experiment with random walk policy with the Gradient Monte Carlo, Semi-Gradient TD (0) and exact value function methods, the exact value function is the most accurate, showing precise values as expected. The other two methods approximate the exact value function with varying degrees of accuracy with TD (0) being more sensitive to immediate rewards and transitions. Both methods capture the general trend successfully but differ in how they estimate and update the values across the grid.

# CONCLUSION

In this project, we conducted experiments using two fundamental reinforcement learning algorithms—Sarsa and Q-learning—in a grid world environment, as well as Gradient Monte Carlo and Semi-Gradient TD (0) methods in a random walk scenario. The purpose was to evaluate the performance of these algorithms in terms of learning optimal policies and estimating value functions under different conditions.

From first experiment, Q-learning showed a more stable learning process with fewer significant dips in rewards, making it more effective at finding optimal paths and avoiding penalties compared to Sarsa. Sarsa produced safer but sometimes suboptimal paths due to its on-policy nature, while Q-learning discovered more direct and optimal routes by focusing on the best possible outcomes.

From second experiment, Gradient Monte Carlo provided a smooth, averaged estimate of the value function, while Semi-Gradient TD (0) produced sharper gradients due to its sensitivity to immediate rewards. Both methods approximated the exact value function reasonably well, with TD (0) being more sensitive and Monte Carlo offering a broader perspective.

The experiments conducted in this report provide valuable insights into the behavior and effectiveness of different reinforcement learning algorithms. Sarsa and Q-learning, while both effective, showed differences in their approach to policy learning. Similarly, the Gradient Monte Carlo and Semi-Gradient TD (0) methods provided useful approximations of the value function, each with its own strengths and weaknesses.