**Reinforcement Learning**

**Assignment 3**

**Maurico and Michael Bimal**



INTRODUCTION

Reinforcement Learning is a crucial field in artificial intelligence, concentrating on how agents can optimize cumulative rewards through specific actions within an environment. Temporal Difference (TD) learning methods like Sarsa and Q-learning are especially notable for their capability to learn despite having incomplete knowledge of the environment. We explore a grid world scenario, a widely utilized setup in RL studies. This scenario features a finite, two-dimensional grid where an agent must navigate from an initial position to a terminal state, while avoiding designated penalty areas.

Sarsa: a well-known reinforcement learning algorithm, is employed to optimize a policy that maximizes expected cumulative rewards for an agent engaging with its environment. It is an on-policy learning method, which implies it evaluates the policy it actively adheres to, maintaining a Q-function Q(s, a) that forecasts expected cumulative rewards. This algorithm frequently utilizes an episilon-greedy policy to effectively balance the needs of exploration and exploitation.

Q-learning: Another prominent algorithm in reinforcement learning and operates as an off-policy Temporal Difference (TD) learning method. This means it determines the value of the optimal policy without depending on the agent's current actions, even if the agent is following a different policy at the time. Similar to other methods, Q-learning employs an epsilon-greedy policy for choosing actions, which helps maintain a balance between exploring new possibilities and exploiting known strategies.

The Gradient Monte Carlo Method: It is arguably the most widely used algorithm in reinforcement learning, estimating the value of a state by averaging the returns from numerous episodes originating from that state. This approach depends on complete episodes to determine the value function and does not necessitate a model of the environment.

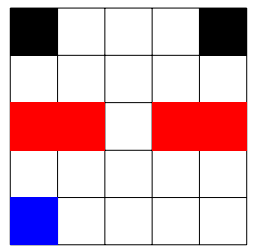
Semi-Gradient TD (0): A temporal difference learning approach that updates the value function using the difference between the current state’s estimated value and the value of the next state, weighted by a linear function approximation. This method combines the aspects of Monte Carlo and dynamic programming.

Objective: This project investigates the application of two fundamental reinforcement learning algorithms, Sarsa and Q-learning, within a grid world framework.

Grid world: The agent starts at the blue square and moves to a neighboring 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 states.

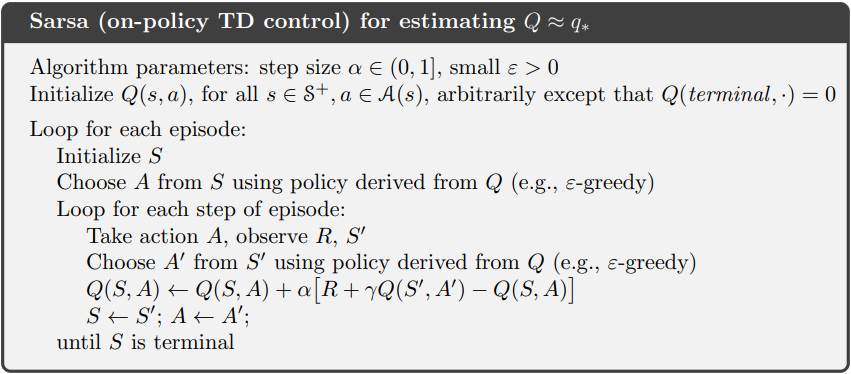
**Part 1**

Consider the following grid world problem



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 states. 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. Use the Sarsa and Q-learning algorithms to learn the optimal policy for this task. Plot a trajectory of an agent utlizing 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.

**SARSA**



*Code explanation*

First, we created a matrix of 5 x 5 and a starting value of 0. We are going to update the value function. The Learning rate α is 0.10 and the discount factor γ is 0.95.



With the function Reward\_And\_Transition\_Part\_1, we will get the reward for each action, and we are going to get the next state. Here, we verify if the agent is in a terminal state, out of boundaries, or in a red or blue cell.

A computer screen with many colorful text

Description automatically generated

The function Next\_Action\_EGreedy will take the following action, verifying that the agent is not outside boundaries. For the SARSA algorithm, the agent takes random actions, so we only have to set a 1 in the epsilon value, and it will take a random action.

A black background with white text

Description automatically generated

We create the loop to update the values for Q\_SARSA, set the starting point (4,0), take the first action, the next position, and the Reward, and take the action and the future position as the pseudocode says. For each episode, the agent will move until the Reward equals 0, which means that it has reached a final state.

A computer code on a black background

Description automatically generated

The following matrix shows the final policy the agent has gotten.

A white background with black text

Description automatically generated

Here are the values obtained for the value function and a heatmap for better visualization.

A black background with numbers

Description automatically generated

A chart with different colors

Description automatically generated

**Q-Learning**

A screenshot of a computer program

Description automatically generated

*Code explanation*

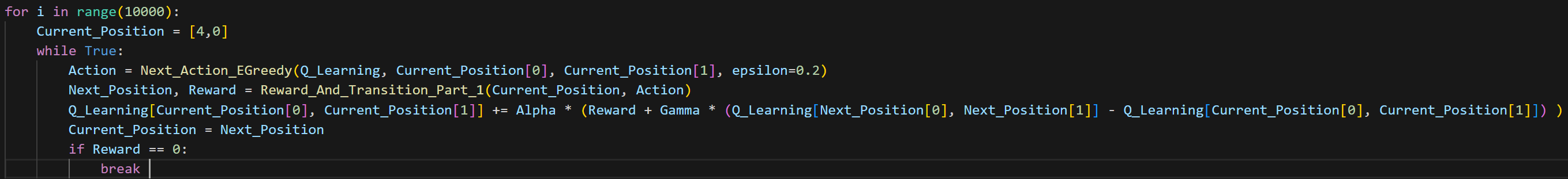
The Q-Learning algorithm is mostly the same as the SARSA algorithm, with the difference that the Q value for the next state should be the highest. We use the same functions as the SARSA algorithm.

A screen shot of a computer program

Description automatically generated

We use the function np.argmax to get the highest value index and take that one.

The agent starts in the state (4,0) and starts taking an action and taking the next state based on the max value; we set an epsilon of 0.10.



The following matrix shows the final policy the agent has gotten.

A grid of black and white text

Description automatically generated

Here are the values obtained for the value function and a heatmap for better visualization.

A black background with numbers and numbers

Description automatically generated

A chart of a graph

Description automatically generated with medium confidence

Algorithm parameters

Step Size α: This is the learning rate, between 0 and 1. A higher α makes the learning updates more aggressive.

Discount Factor γ: This parameter discounts the future rewards as part of the current estimate. It balances the importance of immediate and future rewards, where a value close to 1 gives importance to future rewards, while a value close to 0 makes the agent myopic.

Exploration Rate ϵ: This is used for the ϵ-greedy policy, where ϵ is the probability of choosing a random action. This facilitates exploration of the state space instead of always exploiting the best-known action.