AIL 722: Assignment 2 Report

Anamitra Singha
IIT Delhi

October 2024

Contents

1 Introduction **2**

2 Model-Based Methods **2**
2.1 Policy Iteration 2
2.1.1 Algorithm and Mathematical Explanation 2
2.1.2 Implementation 2
2.1.3 Visualization 3
2.2 Value Iteration 3
2.2.1 Algorithm and Mathematical Explanation 3
2.2.2 Implementation 3
2.2.3 Visualization 4

3 Model-Free Methods **4**
3.1 SARSA and Q-Learning 4
3.1.1 Algorithm and Mathematical Explanation 4
3.1.2 Implementation 5
3.1.3 Results and Visualizations 5
3.2 Evaluation and Analysis 7

4 Large-and-Continuous State-Space Environments **8**
4.1 LunarLander-v2 8
4.1.1 Algorithm and Neural Network Architecture 8
4.1.2 Results and Visualization 9

1 Introduction

This report presents the implementation of several reinforcement learning algorithms, applied to environments such as *TreasureHunt-v1*, *Taxi-v3*, *LunarLander-v2*, and *TreasureHunt-v2*. The report will cover the theoretical background, implementation details, and the performance of these algorithms, followed by visualizations and observations.

2 Model-Based Methods

2.1 Policy Iteration

2.1.1 Algorithm and Mathematical Explanation

Policy Iteration alternates between policy evaluation and policy improvement. The goal is to find the optimal policy π^* by improving a policy π iteratively.

The Bellman equation for policy evaluation:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V^\pi(s')\right]$$

where:

- $V^\pi(s)$: Value function for policy π
- $P(s'|s,a)$: Transition probability
- $R(s,a,s')$: Reward function
- γ: Discount factor

2.1.2 Implementation

The Policy Iteration algorithm was implemented using Python. The environment used is *TreasureHunt-v1*, which consists of 400 states.

2.1.3 Visualization

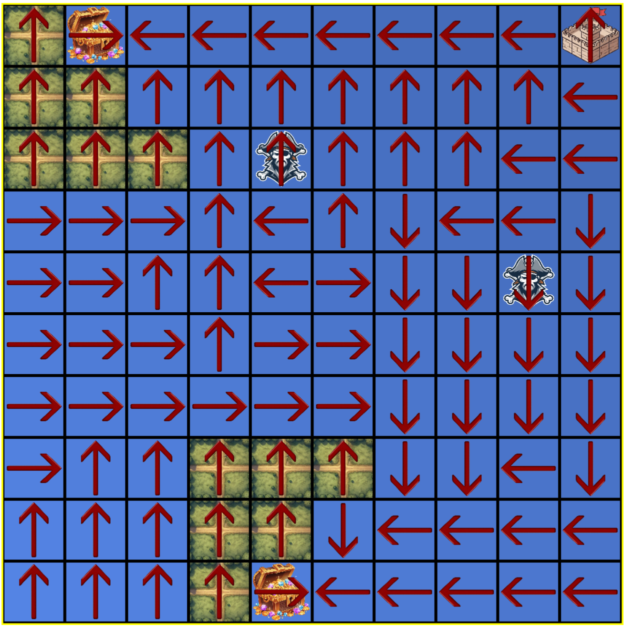

Figure 1: Policy Visualization on the *TreasureHunt-v1* Grid (Arrows represent actions in each state).

Figure 2: Trajectory Visualization of Policy Iteration (GIF)

2.2 Value Iteration

2.2.1 Algorithm and Mathematical Explanation

Value Iteration is based on iteratively updating the value function using the Bellman optimality equation:

$$V(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right]$$

2.2.2 Implementation

The Value Iteration algorithm was implemented, and it was applied to the *TreasureHunt-v1* environment.

2.2.3 Visualization

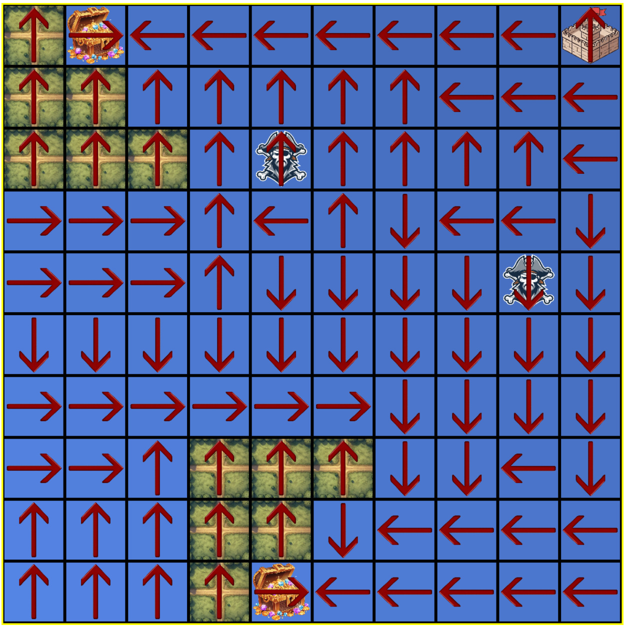

Figure 3: Value Function Visualization on *TreasureHunt-v1*.

3 Model-Free Methods

3.1 SARSA and Q-Learning

3.1.1 Algorithm and Mathematical Explanation

SARSA is an on-policy algorithm that updates the Q-values as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a)\right]$$

Q-Learning is an off-policy algorithm that updates the Q-values using:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Where:

- α: Learning rate
- γ: Discount factor
- r: Reward

3.1.2 Implementation

Both SARSA and Q-Learning were implemented for *TreasureHunt-v1* and *Taxi-v3*. For *TreasureHunt-v1*, the learning rate $\alpha = 0.3$, discount factor $\gamma = 0.95$, and $\epsilon = 0.4$. For *Taxi-v3*, $\alpha = 0.05$ and a decaying ϵ was used.

3.1.3 Results and Visualizations

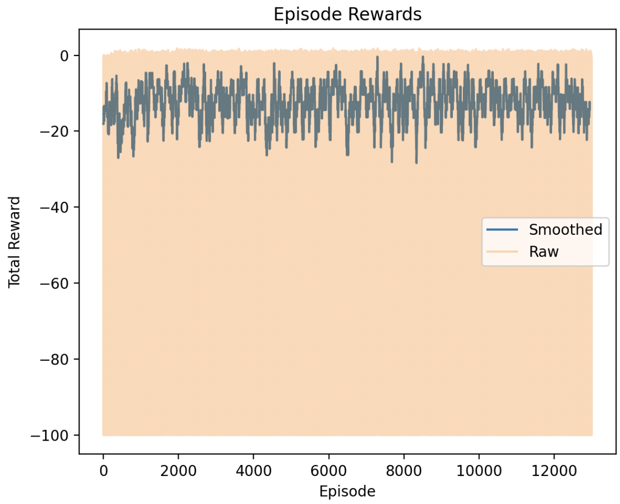

Figure 4: Reward vs Episodes for SARSA (TreasureHunt-v1).

Figure 5: Reward vs Episodes for Q-Learning (TreasureHunt-v1).

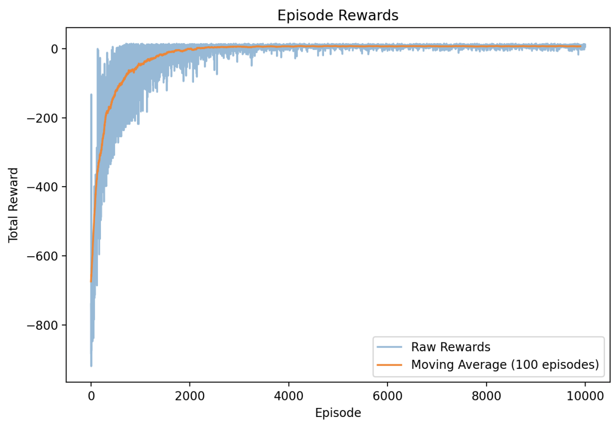

Figure 6: Epsilon vs Episodes for SARSA (Taxi-v3).

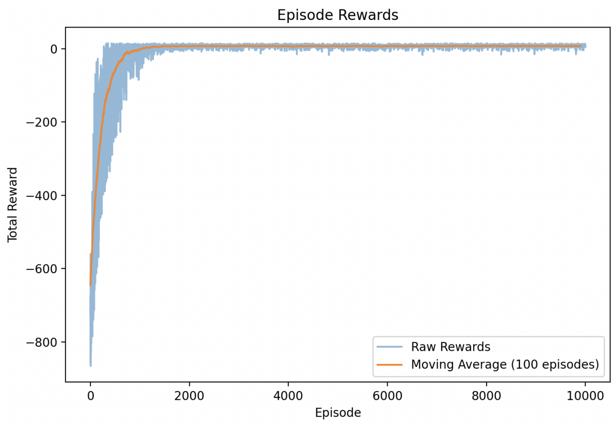

Figure 7: Epsilon vs Episodes for Q-Learning (Taxi-v3).

3.2 Evaluation and Analysis

For *TreasureHunt-v1* the time taken for convergence and the number of episodes were recorded, as shown in Table

Table 1: Comparison of SARSA and Q-Learning Convergence

Algorithm	Episodes for Convergence	CPU Time (s)
SARSA	13000	12
Q-Learning	10000	6

For *Taxi-v3*, the time taken for convergence and the number of episodes were recorded, as shown in Table

Table 2: Comparison of SARSA and Q-Learning Convergence

Algorithm	Episodes for Convergence	CPU Time (s)
SARSA	10000	10
Q-Learning	10000	6

4 Large-and-Continuous State-Space Environments

4.1 LunarLander-v2

4.1.1 Algorithm and Neural Network Architecture

The Q-network used consists of four fully connected layers. The architecture is as follows:

- Input layer: The input state has a size of `state_size`.
- Hidden layer 1: Fully connected layer with 32 units, followed by a ReLU activation:

$$\text{fc1}(x) = \text{ReLU}(\mathbf{W_1}x + \mathbf{b_1}) \quad \text{where} \quad \mathbf{W_1} \in \mathbb{R}^{32\times\text{state\_size}}$$

- Hidden layer 2: Fully connected layer with 64 units, followed by a ReLU activation:

$$\text{fc2}(x) = \text{ReLU}(\mathbf{W_2}x + \mathbf{b_2}) \quad \text{where} \quad \mathbf{W_2} \in \mathbb{R}^{64\times 32}$$

- Hidden layer 3: Fully connected layer with 64 units, followed by a ReLU activation:

$$\text{fc3}(x) = \text{ReLU}(\mathbf{W_3}x + \mathbf{b_3}) \quad \text{where} \quad \mathbf{W_3} \in \mathbb{R}^{64\times 64}$$

- Output layer: Fully connected layer with `action_size` outputs, representing Q-values for each action:

$$\text{fc4}(x) = \mathbf{W_4}x + \mathbf{b_4} \quad \text{where} \quad \mathbf{W_4} \in \mathbb{R}^{\text{action\_size}\times 64}$$

Each layer uses the ReLU activation function except for the output layer, which outputs the raw Q-values for the given state-action pairs.

4.1.2 Results and Visualization

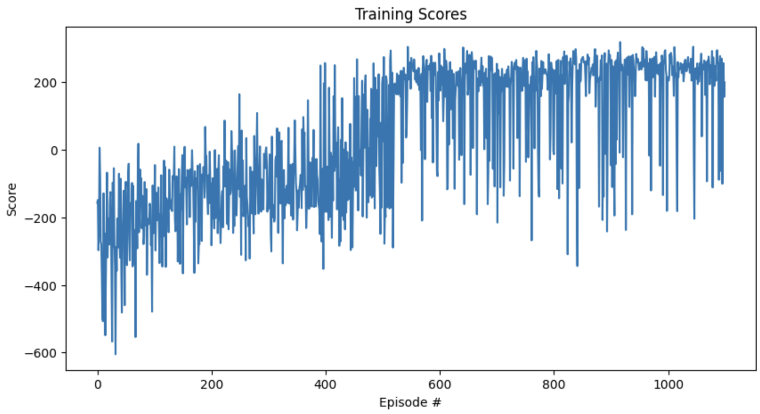

Figure 8: Reward Curve for *LunarLander-v2*.