WiDS ENDTERM REPORT:

By: Kahan Atara

My main motive to participate in this WiDS project was to utilize my winter vacation time and channel it into something which could help me in the future. This project promised just that.

Reinforcement learning is a learning framework in which an agent interacts with an environment by taking actions, receives scalar rewards, and learns a policy that maximizes expected cumulative reward over time through trial and error, without being told the correct actions.

For this WIDS, we followed the books Sutton & Barto: Reinforcement Learning & Grokking: Deep Reinforcement Learning.

Week-1:

The first week of this WiDS project was about learning python and about different directories of python such as:

1. NumPy
2. Matplotlib
3. Pandas

This was crucial as the whole WiDS project was to be coded in python programing language. Mostly, the crucial data types and structures were necessary, yet it was interesting to learn about python in depth.

- NumPy (Numerical Python) is a fundamental library for linear algebra in Python. It plays a crucial role in AI/ML and data science by enabling efficient mathematical and logical operations on arrays.

- Matplotlib is Python's essential plotting library, widely used for data visualization. It supports various types of plots and is the foundation for other visualization packages like pandas and Seaborn

- Pandas is a Python library used for working with datasets. It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool. Pandas has data structures for data analysis. The most commonly used data structures are Series and DataFrame. Series is one-dimensional. It consists of one column. DataFrame is two dimensional. It consists of rows and columns.

Also, we were introduced to GitHub and jupyter notebooks. Although there was no clear-cut way to learn them, we still managed to use these tools for the project.

Week-2:

This week was focused on understanding value function and policies, and implementing algorithms on the Multi-Armed Bandit Problem

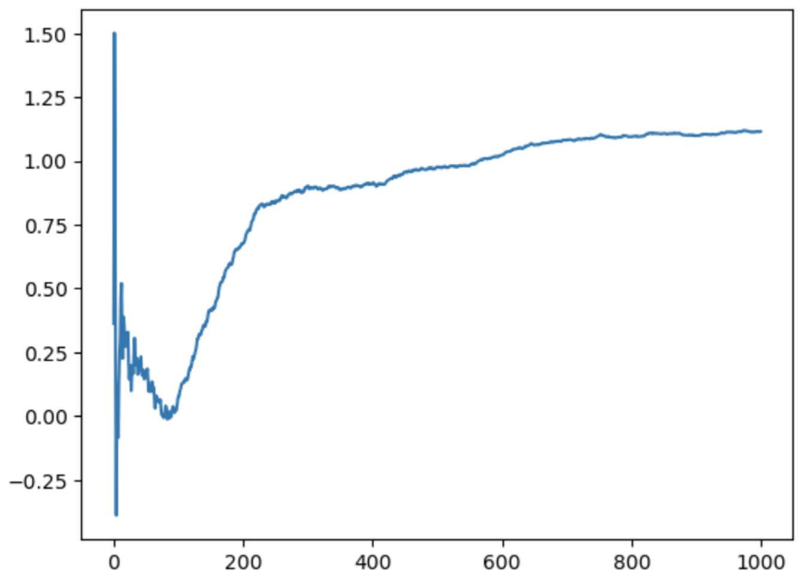
Policy:- A policy π is a mapping from states to a probability distribution over actions:

Value function:- The value function of a policy π is the expected return starting from state s and thereafter following π

The multi-armed bandit problem models an agent facing a set of k actions (arms), each producing stochastic rewards from a Normal distribution with unknown mean. At each time step, the agent chooses one arm and observes its reward, with the objective of maximizing cumulative reward over time. In our version, there are 10 arms.

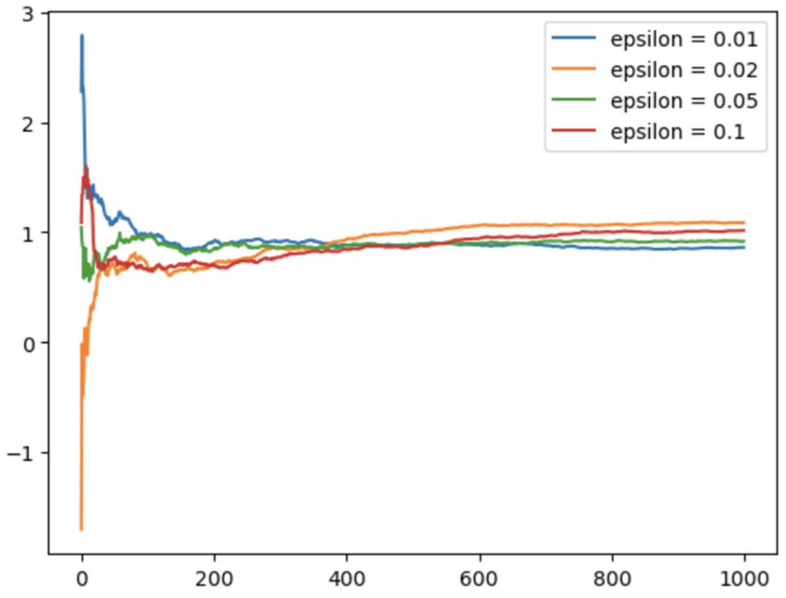
We implemented the following algorithms

1. Greedy Algorithm:- Always selects the action with the highest estimated value so far. It exploits current knowledge but does **no exploration**

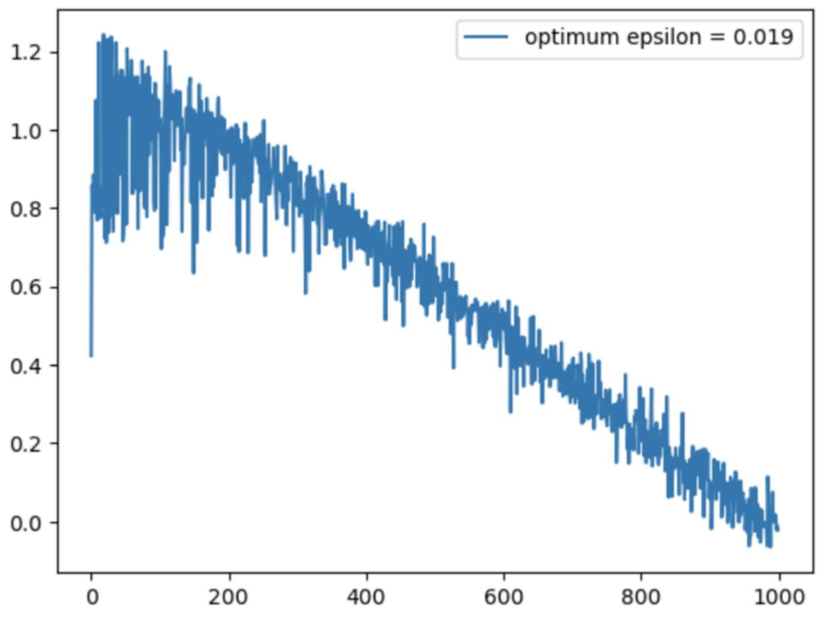


(Graph for greedy algorithm implementation)

2. Epsilon Greedy Algorithm:- With probability 1−ε, chooses the greedy (best-known) action, and with probability ε, selects a random action. This has both exploration and exploitation

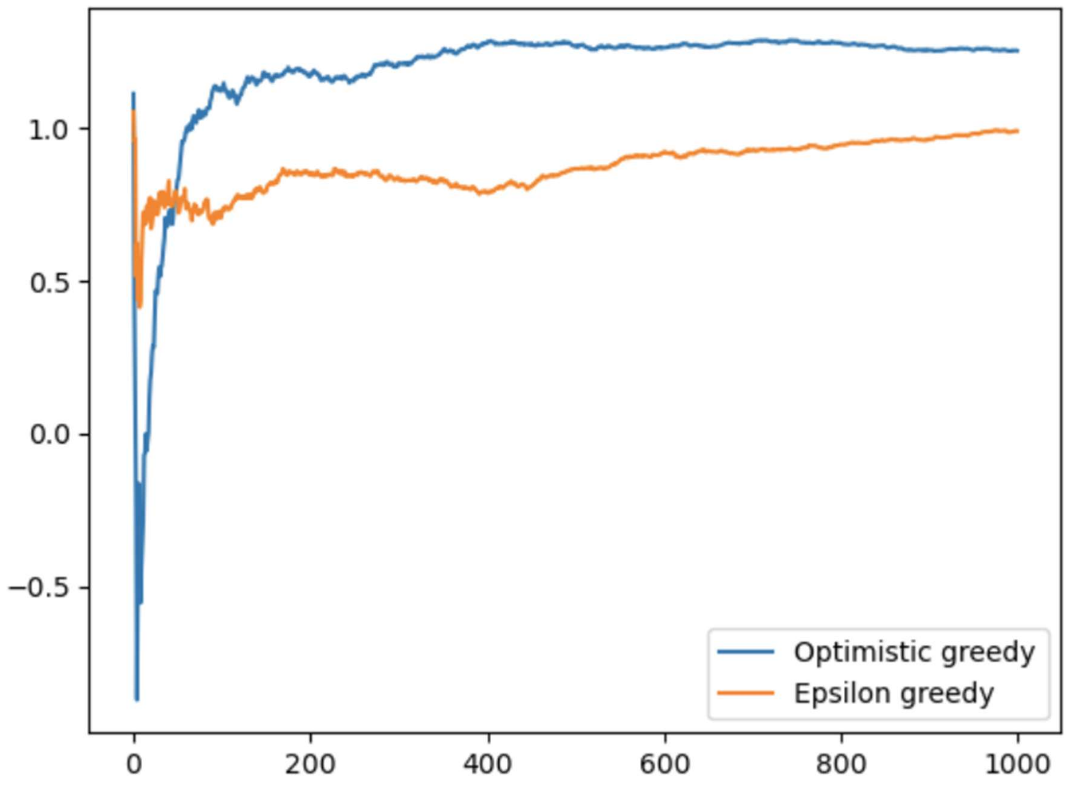


(Graph for epsilon-greedy algorithm implementation)



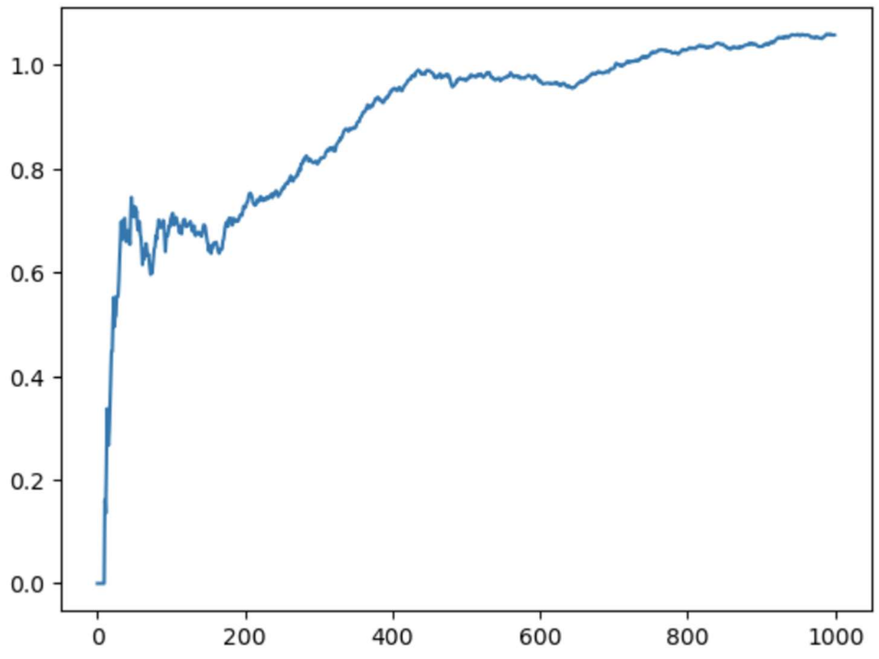
(Graph for optimum epsilon value)

3. Optimistic Initial Values:- Initialize all action-value estimates to unrealistically high values. Early actions tend to look bad after being tried once, which naturally encourages exploration without explicit randomness



(Graph for comparison between Optimistic greedy algorithm and Epsilon greedy algorithm)

4. Upper Confidence Bound:- Selects actions based on both estimated value and uncertainty, choosing the action that maximizes Q(a) + c * root(ln(t) / N(a))



(Graph for Upper Confidence Bound algorithm)

Week-3:

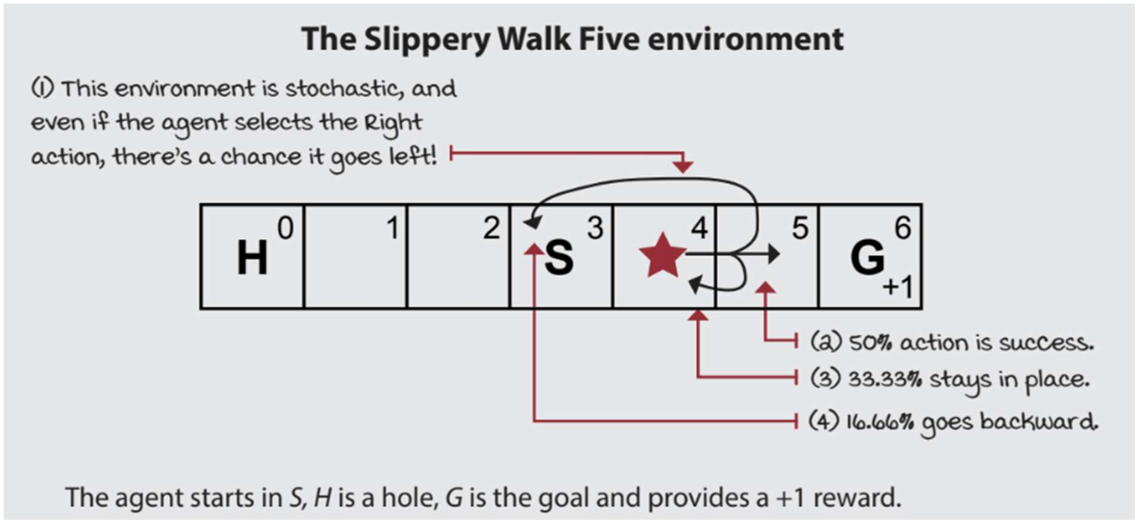
This week consisted of understanding Markov decision processes and coding them and also understanding Bellman Optimality Equations

A Markov Decision Process (MDP) is a mathematical framework used to model sequential decision-making where outcomes are partly random and partly controlled by an agent. MDPs define an agent interacting with an environment over discrete time steps to maximize long-term rewards, adhering to the Markov property where future states depend only on the current state and action. An MDP consists of the 5 parts

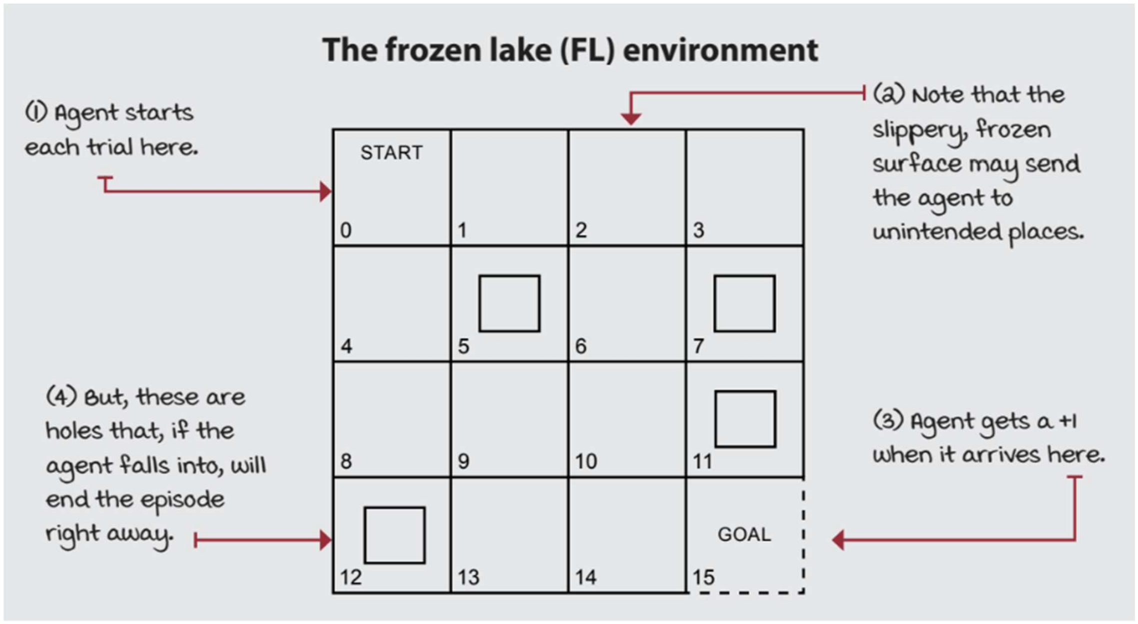
1. S = Set of all possible states
2. A = Set of all possible actions
3. P = transition probability function
4. R = reward function
5. Gamma = discount factor

After this, we coded in various sample MDPs, which were

1. Slippery Walk


The Slippery Walk Five environment
(1) This environment is stochastic, and even if the agent selects the Right action, there's a chance it goes left!
H 0
1
2
S 3
4
5
G 6 +1
(2) 50% action is success.
(3) 33.33% stays in place.
(4) 16.66% goes backward.
The agent starts in S, H is a hole, G is the goal and provides a +1 reward.

2. Frozen Lake


The frozen lake (FL) environment
(1) Agent starts each trial here.
(2) Note that the slippery, frozen surface may send the agent to unintended places.
(3) Agent gets a +1 when it arrives here.
(4) But, these are holes that, if the agent falls into, will end the episode right away.
START
GOAL
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

We also studied Bellman optimality equations, which allow us to iteratively compute the optimal value functions.

Bellman Equation for value functions,

Bellman Equation for v_π:

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{\substack{s' \in \mathcal{S} \\ r \in \mathcal{R}}} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Bellman Equation for action value functions,

Bellman Equation for q_π:

$$q_\pi(s, a) = \sum_{\substack{s' \in \mathcal{S} \\ r \in \mathcal{R}}} p(s', r|s, a)\left[r + \gamma \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q_\pi(s', a')\right]$$

Week-4:

This week was implementation of our learnings about core reinforcement learning so far for a stock trading task.

In this project, I implemented a complete RL pipeline for algorithmic trading. The main components included:

- A **custom trading environment** that simulates market interaction using historical stock price data.
- A **tabular Q-learning agent** that learns an optimal policy over discrete states and actions.
- An **evaluation framework** to test the trained agent on unseen data.
- A **baseline comparison** with a Buy-and-Hold strategy to contextualize the agent's performance.

This notebook serves as a foundation for more advanced extensions such as Deep Reinforcement Learning (e.g., DQN) and the inclusion of technical indicators.

Data Collection and Preprocessing

I used historical stock price data downloaded through the yfinance library. The data was split into training and testing periods to ensure a fair evaluation. Prices were processed sequentially to simulate real-time trading.

Trading Environment

I designed a custom environment that tracks:

- Current time step
- Stock price at that step
- Agent's position (holding or not holding a stock)
- Cash balance and portfolio value

At each step, the agent could choose one of three actions: **Buy**, **Sell**, or **Hold**. The reward was defined based on changes in portfolio value, encouraging the agent to make profitable decisions.

Q-learning Agent

The agent used a tabular Q-learning approach where:

- States were discretized representations of market and portfolio conditions.
- Actions corresponded to Buy, Sell, and Hold.
- Q-values were updated using the standard Q-learning update rule.

Hyperparameters such as learning rate (α), discount factor (γ), and exploration rate (ε) were tuned experimentally to achieve stable learning.

Training and Evaluation

The agent was trained over multiple episodes on historical data. After training, performance was evaluated on unseen test data. I plotted portfolio value over time and compared it with a Buy-and-Hold strategy to analyze relative performance.

Conclusion

This project was a valuable learning experience that solidified my understanding of Reinforcement Learning by applying it to a practical trading problem. While the tabular Q-learning agent has limitations, it provided a strong foundation for understanding more advanced approaches such as Deep RL. Overall, this assignment successfully demonstrated how RL concepts can be applied to trading and prepared me for further exploration in this domain.