Winter in Data Science (WiDS 2024-25)

Medhansh Verma
First year, CSE Department

February 10, 2025

Introduction

I have successfully completed the WiDS project (UID 63) titled **"Can Computers Think?"**. Below is my GitHub repository for reference:
GitHub Repository

This document summarizes my weekly progress and key learnings throughout the project.

Week 0: Foundational Concepts

We were provided with four extensive Jupyter notebooks covering **NumPy, Pandas, Matplotlib, and Python**. Although these notebooks only scratched the surface of Python, they helped build a solid foundation, allowing me to approach the project with ease.

Week 1: Multi-Armed Bandit Problem

This week, we began focusing on problems from *Sutton and Barto*. My assigned problem was the **Multi-Armed Bandit** problem. A provided class `bandit.py` simulated the levers of bandits. I implemented the following RL algorithms:

- Greedy Algorithm
- ε-Greedy Algorithm

- Optimistic Initial Values
- Upper Confidence Bound (UCB) Algorithm

The basic thought process was to maintain a running count of each bandit's average reward. With enough iterations, we could estimate each bandit's mean and standard deviation, thus improving the average reward per pull.

Week 2: Markov Decision Processes (MDPs)

This week, we focused on Markov Chains and creating Markov Decision Processes (MDPs). Key takeaways:

- MDPs are essentially a dictionary of dictionaries storing state-action-reward-next state data.
- Handwriting small MDPs is feasible, but automation is crucial (and fun!).
- MDPs model the environment, allowing an agent to determine the best **policy** (covered in Week 3).

Week 3: Policy and Value Iteration (Jack's Car Rental and Gambler's problem)

This was the most intellectually stimulating week for me. **Jack's Car Rental**, from Sutton and Barto, was an excellent exercise in policy iteration. I implemented:

Policy Iteration

1. Initialize a policy and value function.
2. **Policy Evaluation**: Iterate over all states and update their values based on the Bellman expectation equation.
3. **Policy Improvement**: Update the policy by selecting the action that maximizes expected return.
4. Repeat until the policy converges.

Value Iteration

- Directly updates the value function, extracting the optimal policy at the end.
- Faster per iteration but may require more iterations overall.

Key Findings:

- **Policy Iteration**: Fewer policy changes lead to faster convergence despite expensive evaluations.
- **Value Iteration**: Suitable for small state-action spaces where quick value updates are possible.

> Bellman Equations
>
> For more details on the Bellman equations used, check out my Markdown documentation in the repository.

Weeks 4-5: Final Project - Solving the 15-Puzzle

The final project involved solving the **15-Puzzle**, which proved to be the most challenging yet exciting part of WiDS.

Approach:

- The puzzle is a 4x4 grid presented to the agent as a list, with 16 denoting the empty cell.
- Given the enormous state space (16!), I adopted a row-wise solving method.
- Solved rows were masked with `-1` (immovable), while unnecessary rows were marked with `0`.
- This significantly reduced the state space, enabling convergence in approximately 4.5 minutes.
- The final segment of code generates a random puzzle and solves it in atmost 80, which is pretty good for a row-oriented solution.

Observing how the RL agent efficiently positions the last row, simultaneously solving rows 3 and 4 after fixing the first two, was fascinating.

This project taught me a lot: from debugging and refining my policy and reward function to efficiently masking and visualizing the puzzle. I have gained immense confidence in **Python and Reinforcement Learning**.

Acknowledgment

This project was offered by the **Analytics Club, IIT Bombay**

I am grateful to my fabulous mentors: **Nirav Bhattad and Balaji Karedla**. Their guidance was invaluable in shaping my understanding and problem-solving skills.

Thank you!