WiDS 2024: Can computers think?

Advait Gupta

February 2025

Abstract

This project was to explore the basics of Reinforcement learning, with a structured approach to understanding its core concepts.
The focus was on studying theoretical aspects and implementing fundamental algorithms. We primarily used Sutton and Barto as a reference and our learning was focused on the first 4 chapters of the book namely:

1. Introduction
2. Multi armed Bandits
3. Finite Markov Decision Processes(Although for this part we didn't refer to this book and instead referred to the book by Miguel Morales.)
4. Dynamic Programming

For each phase, we were given reading assignments from books and assignments similar to the book examples which helped in clarifying concepts and learn implementations as we coded them ourselves.
For the final project we had solve the 15 puzzle using Reinforcement Learning. The project was mostly just a clever implementation of the previous 2 phases of material.
In this report I will first give some intro about RL, and after that I will go phase-wise telling what I did, briefly what I learnt, and the results of what I did wherever applicable.

1 Introduction

Here I will be giving some basic intro about RL, involving stuff I learnt while reading the chapter 1 of Sutton and Barto before the project had begun. It is not directly linked to my work on the project. Feel free to skip this part.

The idea of learning via interaction with our Environmental is a very natural one. An interesting example for this is babies learning how to walk. They essentially just try to do it, and get clear reward/punishment feedback(in terms of walking vs falling down) and using that they eventually learn how to walk. This idea of trial and error and learning via interaction with environment is fundamental to Reinforcement Learning.

Reinforcement learning is learning **what action to take situation wise** so as to **maximize a numerical reward signal**. The learner is not told which actions to take, but instead must **discover which actions yield the most reward by trying them**. In most cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards.
One of the challenges that often arises in Reinforcement Learning is the **trade off between exploration and exploitation**. As our purpose is to maximize the reward, we naturally need to take greedy actions often i.e. actions that we have identified to yield good reward. But we also need to do exploration, as we might find better actions. A lot of discussion has been done on this part in the Multi Arm Bandit section(phase 2 of our project).

Elements of Reinforcement Learning:

- **Policy**: It is essentially a mapping between the states of the environment and the action that the Learning agent will take in that state.
- **Reward Signal**: The reward number environment sends to the agent after each step.
- **Value function**: Roughly, it is the expected value of total amount of reward the agent will accumulate starting from that state.
- **Model**: Mimics behaviour of environment.(Unlike previous parts, this one is optional)

2 Phase 0: Learning Basic tools

Initially we were given some time to learn Python, numpy, matplotlib, pandas. We were given some recommended resources and helping Jupyter notebooks for learning. This proved to be extremely useful throughout the project(and even beyond that) as Python and numpy were used for all of the coding part and matplotlib was extremely useful in data visualization for comparing different algorithms etc.

3 Phase 1: Multi Arm Bandit

We were given a reading assignment to read Chapter 2 from Sutton and Barto which is based on the Multi armed bandit problem.
The problem:
You are faced repeatedly with a choice among k different options. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your object is of course to

maximize the total expected reward after some time period.
Now, each of the k options must be having some expected reward, we call that **the value of that action**. However, note that these values are not known to us. If they were know to us, we would simply select the highest value actions each time. Hence, after each step, we will keep an estimate of the values for all the options.
And here we will encounter the Exploitation vs Exploration problem I described in the introduction. Among our list of estimated values, there will be at least one highest value and we call the corresponding action, a greedy action. **Taking the greedy option is exploitation and taking anything else is exploration** and helps in better estimating the values of currently non-greedy options(they could very well turn out to be better than the current greedy option eventually). Balancing between exploration and exploitation is a pretty sophisticated problem, as it depends on a variety of factors such as the values themselves, number of remaining steps etc.
Estimating action values:
A pretty intuitive formula is used: $\frac{\text{sum of rewards obtained when that action taken previously}}{\text{number of times that action has been taken}}$
However for a non-stationary problem we use an incremental update rule for estimation:

$$Q_{n+1} \doteq Q_n + \alpha \left[R_n - Q_n \right]$$

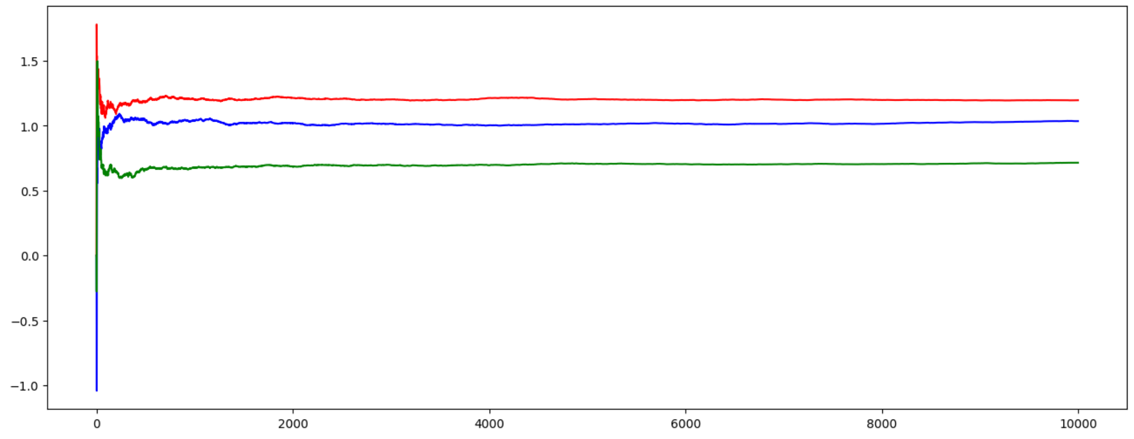
where:

- Q_n refers to the estimate after $n-1$ steps.
- R_n refers to the reward obtained in the nth step.
- α is a constant belonging to $(0, 1]$.

Now, in the coding assignment we implemented various methods and algorithms in this problem. I will write a brief explanation of each along with my results after this.

- Applied the greedy, ϵ-greedy algorithms and compared the results with a graph:

  Greedy: The greedy algorithm is simply selecting the greedy action each time.
  ϵ-greedy: In ϵ-greedy algorithm, we behave greedily most of the time, but once in a while with probability ϵ, we select an action randomly out of all the options with equal probability for the purpose of exploration.

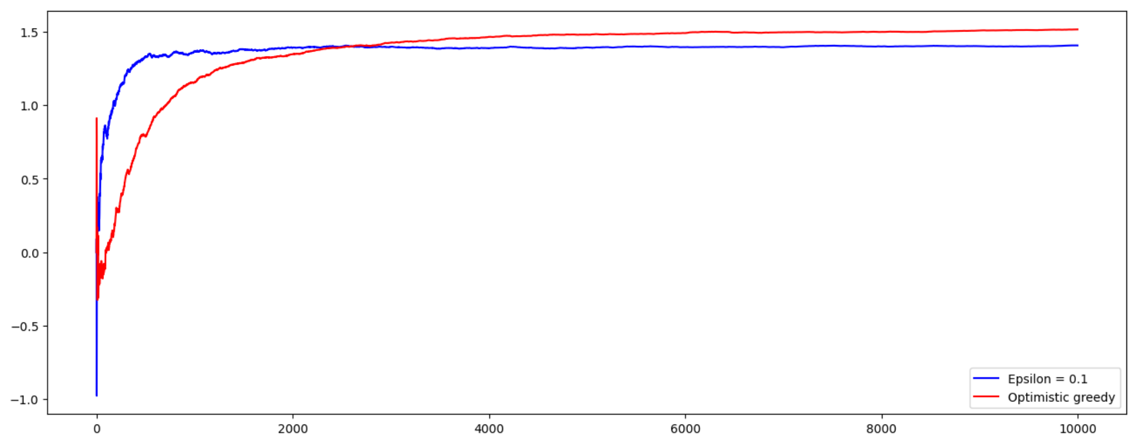


This is a plot of variation of the cumulative average of rewards as iterations increase.
Here Red graph is for ϵ=0.01
Blue graph is for ϵ=0.1
Green graph is for Greedy.

My inference:
Clearly ϵ-greedy algorithm works better soon because of more exploration. However **it is worth noting that in the long term, low ϵ algorithm works better**. This is because after a large number of iterations for a non-zero ϵ, we will have good estimates of action values. Beyond this, greedy is good and exploration is pretty much unnecessary. Having a higher ϵ at this point leads to unnecessary exploration reducing the cumulative reward. Hence it is better to reduce ϵ as number of iterations increase.

- Optimistic Initial Values method:
  In the methods we discussed, there is some initial value bias. For the sample-average methods, the bias disappears once all actions have been selected at least once, but for methods with constant α, the bias is permanent.
  We can use the initial action values to encourage exploration. Suppose instead of setting the values to zero, we set them to a high value say +5 which is highly optimistic(reward is between 0 and 1). Then whenever an action is tried, disappointment is faced and more exploration occurs. Hence in this method, even if we use a greedy strategy there will be a fair bit of exploration.

  I implemented an optimistic greedy with initial values of 10 and compared it with ϵ-greedy with ϵ=0.1.



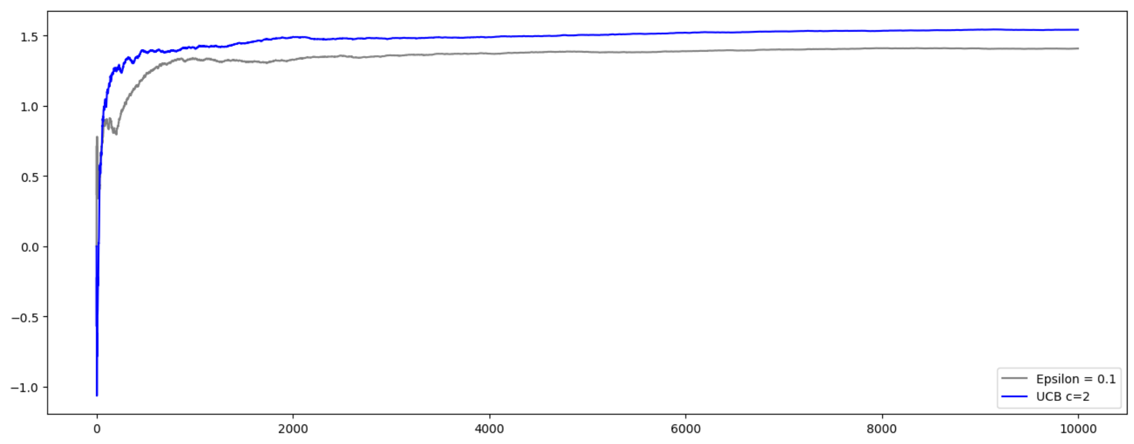
This is plot of cumulative average of rewards as iterations increase.

Clearly initially for a long time, epslion beats it but in the long run optimistic greedy wins as both have done proper exploration. Optimistic greedy always selects greedily and cashes in it's exploration while epsilon still often unnecessarily goes for exploration reducing the cumulative reward.

- Upper Confidence Bound:
  ϵ-greedy action selection forces the non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain. **It would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates.** Hence in this method for determining we use the following formula which takes into account both how high is it right now(for greediness) and it's potential for being high(for exploration):

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

I implemented UCB with c=2 and compared it with ϵ-greedy, ϵ=2.



This is plot of cumulative average of rewards as iterations increase.

Clearly, UCB is better. However, it is not always feasible to use UCB in more complicated problems and we often end up not using it.

4 Phase 2: Markov Decision Processes

We were given a reading assignment to read the chapter 2(Mathematical foundations of reinforcement learning) from the book by Miguel Morales. In this we learnt the components of RL, basics about MDPs and how to code MDPs.

Most problems we work with using RL always have a MDP working under the hood, which we may or may not(most of the times) know.

In the coding assignment, we were given 2 problems, one involved hard-coding a straightforward Slippery walk environment MDP.
The second problem of coding Frozen lake environment MDP was slightly more complicated and we were supposed to automate MDP generation for this part(basically create using loops, conditionals and general statements instead of harcoding). This was a nice practice, especially useful in the assignment for the next week and in the final project.

5 Phase 3: Dynamic Programming

In reading assignment, we were told to read the chapter 4 from Sutton and Barto which is on Dynamic Programming.
Policy Evaluation:

Policy evaluation involves computing the state-value function $V^{\pi}(s)$ for a given policy π. This is done iteratively using the Bellman expectation equation:

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} P(s',r|s,a)\left[r + \gamma V_k(s')\right]$$

where $P(s',r|s,a)$ represents the transition probabilities, and γ is the discount factor.

Policy Improvement:

Given a policy π, we can derive a better policy using the policy improvement theorem. A new policy π' is defined by acting greedily with respect to the current state-value function:

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

Policy Iteration:

Once a policy, has been improved to yield a better policy, we can then compute the new policy and improve it again to yield an even better one. We can thus obtain a sequence of monotonically improving policies and value functions. Since a finite MDP has only a finite number of deterministic policies, this process must converge to an optimal policy and the optimal value function in a finite number of iterations.
This way of finding an optimal policy is called policy iteration.

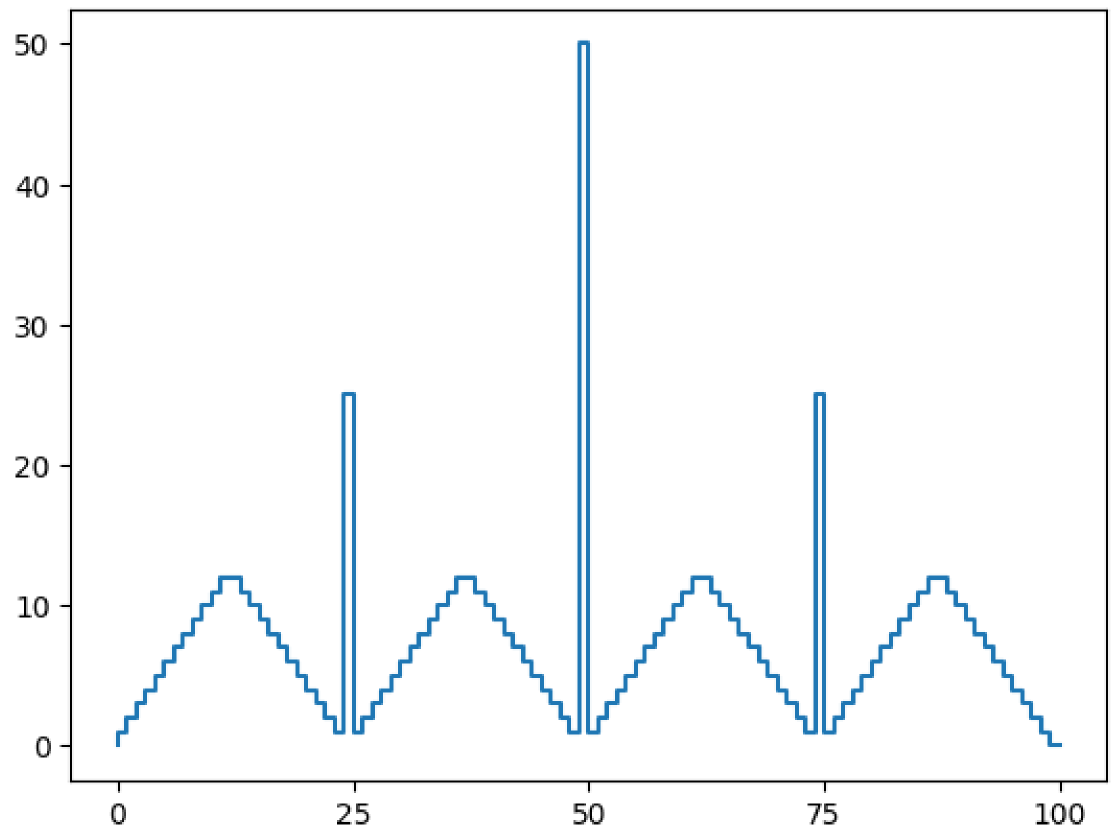
Value Iteration:
One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set.
Instead of fully evaluating a policy before improving it, value iteration updates the value function directly using:

$$V_{k+1}(s) = \max_a \sum_{s',r} P(s',r|s,a)\left[r + \gamma V_k(s')\right].$$

This method is computationally more efficient and often converges faster.

The coding assignment was to implement the Gambler's problem as described in the book.

Here I had to first code up the MDP for the problem which required automation.(This was relatively straightforward after the nice practice in the previous phase).
Then I applied the value iteration method, with gamma=1 to get a policy and plotted the policy as follows(x-axis represents state and y-axis represents betting amount in that state):



It seems it tries to reach target values 25,50,75,100. In the states 0,25,50,75,100 we see that it bets the max possible amount.

6 Phase 4: Final Project

We has to solve the 15-puzzle(click on it to see and play the puzzle) using Dynamic programming.

- First I decided how to characterize the states of the puzzle. For that I decided to use a 16 element array which will be having each of the rows one by one as it's elements. We will represent the empty space by 16.
- Then, developing some high level idea, we realize that the number of states is way too high and directly applying value iterations on all the states will be computationally too expensive. Hence, I decided to solve the puzzle row-wise. That is the first row first, then the second row and the remaining two rows together. Now for each part I don't need to see what's at the remaining places and can essentially just treat them as 0s. It is easy to

see that this drastically reduces the total number of states. However, it is noteworthy that with this we also lose optimality of solution as it is possible that the optimal solution doesn't go row-wise.(But the point of the project is not optimality so it's fine).

- So basically we essentially need to take in input puzzle. First make a new state array from this where everything except 1,2,3,4,16 are 0s. And then I need to apply policy for first row on this, and the same actions I'll also implement on the actual input puzzle. Then the resultant will be sent for part 2 and then to part 3. For policy, we will need different MDPs for each part, and some more details will be needed which can be checked out in my code and are not worth discussing here.

- I precompute the policies in my code as the environment is known, we can before-hand figure out the policies and then simply implement them on the input puzzle the way I told above.

7 Conclusion

Through this project, I gained a nice basic understanding of Reinforcement Learning by studying theoretical concepts and implementing core algorithms and also developed an interest in it and intend to dig deeper into it.
Beginning with Multi-Armed Bandits, I developed an intuition for the exploration-exploitation trade-off and dynamic decision-making.
The DP portion was very intuitive and interesting to get into. Also, the DP assignments and final project were very cool and MDP assignment acted as a nice bridge.
This project helped me learn a lot of stuff not only in RL, but in general. My coding skills drastically improved with the project and while doing this project, I also did a fair bit of exploration of similar topics and developed a strong interest in Machine Learning which I intend to properly pursue in the future.