WiDS Project Report

UID 63: How Machines Learn: A Journey Through Reinforcement Learning

- Kashyap Khandelwal

Following is the report of what I learnt through this project under Reinforcement Learning. We used “Reinforcement Learning: An Introduction” by Sutton & Barto as a reference.

After covering foundational RL concepts, including agents, rewards, policies, and value functions, we moved onto Markov Decision Processes and Policy Evaluation and Improvement. Implemented key Reinforcement Learning (RL) concepts, such as: 10-Armed Bandit Problem, Frozen Lake Environment, Markov Decision Process, Jack’s Car Rental Problem and the Gambler’s Problem. Finally, implemented RL to solve the 15-puzzle problem.

Each week we were given a specific chapter from the reference book to go through, after which we were supposed to attempt problems related to that. We covered the following chapters from the book: Introduction, Multi-Armed Bandits, Finite Markov Decision Processes, Dynamic Programming. We were also given “Grokking Deep Reinforcement Learning” by Miguel Morales as reference for understanding MDPs further.

Week 0:

We learnt basics about python through tutorial videos. Also, learnt some Python libraries such as NumPy, Panda and Matplotlib through Jupyter Notebooks given in the GitHub repository. Matplotlib proved to be a powerful tool to visualize data.

Week 1:

We studied Chapter 2 of the reference book. The problem statement as given in the book:

> *You are faced repeatedly with a choice among k different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your objective is to maximize the expected total reward over some time period, for example, over 1000 action selections, or time steps.*

If you maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest. We call these the greedy actions. When you select one of these actions, we say that you are exploiting your current knowledge of the values of the actions. If instead you select one of the nongreedy actions, then we say you are exploring, because this enables you to improve your estimate of the nongreedy action’s value. Because it is not possible both to explore and to exploit with any single action selection, there is always a “conflict” between exploration and exploitation.

We implemented the Greedy Algorithm to solve the 10-Armed Bandit Problem, where the agent always selects the greedy action at each state. Then, implemented the epsilon-greedy algorithm where there is an epsilon chance that it will explore more actions. The epsilon-greedy methods eventually performed better because they continued to explore and to improve their chances of recognizing the optimal action.

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. Suppose instead of setting the values to zero, we set them to a high value say +10 (which is highly optimistic). 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.

Also implemented the Upper Confidence Bound Action Selection. Epsilon-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. UCB selects 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.

Week 2:

A common way to represent decision-making processes in RL is by representing the problem using a mathematical framework known as Markov decision processes (MDPs). In RL, we assume all environments have an MDP working under the hood.

As part of the assignment, we created the Bandit Slippery-Walk Environment. The Bandit Walk is a simple grid-world environment with three states, but only one non-terminal state. The BW environment has just two actions available: Left and Right. The reward signal is a +1 when landing on the rightmost cell, 0 otherwise. The agent starts in the middle cell. The Slippery Walk introduces stochasticity into the environment where each action has, say, only a 50% chance of sending the agent according to the action and 25% of remaining where it is and 25% of going backwards.

The Frozen Lake environment has a large state space, so we generated most of the MDP via Python instead of typing stuff manually.

Week 3:

After reading the chapter on Dynamic Programming in our reference book, we solved the Jack's Car Rental Problem and the Gambler's Problem.

The basics of DP which we learnt this week:

- Policy Evaluation: Policy evaluation involves computing the state-value function V_π for an arbitrary policy π. This is done iteratively using the Bellman expectation equation:

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

- Policy Improvement: Our reason for computing the value function for a policy is to help find better policies. We can do so by following the policy improvement theorem

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

- Policy Iteration: Once a policy has been improved using to yield a better policy, we can then improve it again to yield an even better policy. We can thus obtain a sequence of monotonically improving policies and value functions. Because 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.

- 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. The policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps. This algorithm is called value iteration

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

Week 4:

Implemented solving the 15-Puzzle Problem using Reinforcement Learning. The 15-puzzle is basically a 4x4 square with 15 movable tiles in it. There is a single tile vacancy in the box, which can be used to shift tiles around. The idea was not to have the machine generate the fastest possible solution, but rather to generate a more intuitive solution, similar to what a human would do if he were solving the puzzle. If you play this puzzle, you would naturally think that it is probably a good idea to start fixing the first row (1, 2, 3, 4) and then the second row (5, 6, 7, 8) and then the third and fourth rows. While you are trying to put 1, 2, 3 and 4 into their places, you probably don't care about the other numbers. They all would look the same to you as if they don't exist and their number doesn't matter. And that is the main idea we use here to reduce the total number of states. Thus, I implemented a phase-wise approach to solve the 15-puzzle. I took around 70 steps to solve it, which isn't too bad!

Conclusion:

This project deepened my understanding of reinforcement learning, from foundational concepts to advanced problem-solving techniques. Implementing various RL algorithms improved my problem-solving skills and strengthened my grasp of key ideas like exploration-exploitation and policy optimization. Solving the 15-Puzzle was particularly rewarding, and overall, this experience has further increased my interest in the field of AI and machine learning.