Hands-On Reinforcement Learning

WiDS, Analytics Club, IIT Bombay

Soham Rahul Inamdar

January 29, 2023

1 Overview of the project

This project was an introductory project in **Reinforcement Learning**, an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

1.1 Project Flow

The project started off with our mentors providing us with a bird's eye view of RL and a review of basic probability theory which forms the very basis of most areas of Machine Learning. This was followed by a study of the n-armed bandit problem and various algorithms to solve it which are described in detail further in this report. Further, we studied the mathematical formulation of Markov Decision Problems(MDPs) and methods of solving them in a rather rigorous fashion. Overall, this was a project with a quite steep learning curve when it came to the mathematical part which helped me understand the basic formulation of reinforcement learning problems and will enable me to explore this field further.

2 Part-1

2.1 Bird's Eye View of Reinforcement Learning

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. The primary difference between reinforcement learning and most other forms of machine learning is that when it comes to RL, the learning agent is not told which actions to take but instead the learner discovers which actions yield the largest reward by trying them out.

2.2 The Multi-Arm Bandit Problem

The original form of the $n - armed\ bandit\ problem$ can be seen as a slot machine having n levers which will be referred to as actions in reference to the problem. So, the agent is supposed to choose an action to maximize the cumulative reward. If the agent already knew the value of each action it would be trivial to solve this problem.

But how do we proceed when the agent has no prior information about the values corresponding to each action?

Solving the above problem requires us to build a mathematical framework that will help us provide elegant solutions without any $hand-wavy$ assumptions. We denote the true value corresponding to an action a by $q(a)$ and the estimated value at the t^{th} time-step to be $Q_t(a)$.
So if the action a has been taken $N_t(a)$ times before time t with corresponding rewards being $R_1, R_2, R_3..., R_{N_t(a)}$:

$$Q_t(a) = \frac{R_1 + R_2 + R_3 + ...R_{N_t(a)}}{N_t(a)} \tag{1}$$

We define the initial value i.e $Q_1(a) = 0$ as a convention.
So, now our goal is to maximize the average reward over time. As a part of this project, we explored three algorithms to achieve this goal which are as follows:

1. **ϵ-greedy Algorithm:** The most obvious idea one can have to solve our problem is to always choose the action with the highest value. However, in this method, the agent will not be able to explore and estimate the values of other actions which may be more than that of the currently optimal action. So the algorithm we follow is that we choose the optimal action $A_t = argmax\, Q_t(a)$ most of the time, thereby exploiting our previous knowledge. However, with a small probability ϵ, we choose an action at random so that we can explore other states as well.

2. **UCB Algorithm:** Owing to the uncertainty in action-value estimates, one needs to explore some non-optimal actions as well as we saw in the ϵ-greedy algorithm. But, instead of choosing the exploratory action randomly, we now take into account how close the estimate of a non-greedy action is to the optimal value and hence the algorithm we follow here is:

$$A_t = argmax\left[Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}}\right] \tag{2}$$

   Here, the factor c controls the degree of exploration. An action a is considered to be a maximizing action if $N_t(a) = 0$. The square-root expression is a measure of the uncertainty in the estimated value of action a.

3. **Thompson Sampling Algorithm:** This is another algorithm for sampling/pulling a MAB. Here, a list of successes $s_t(a)$ and failures $f_t(a)$ of the arms is maintained, and then the following samples are drawn for each arm,

$$n_t(a) = \beta(s_t(a) + 1, f_t(a) + 1) \tag{3}$$

$$A_t = argmax(n_t(a) \tag{4}$$

   where β denotes the Beta Distribution. Then, the arm with the highest value of such a sample is chosen. In the background, a bayesian update is happening, which ensures that this algorithm works well.

The performance of the three algorithms can be seen in the graphs given below:

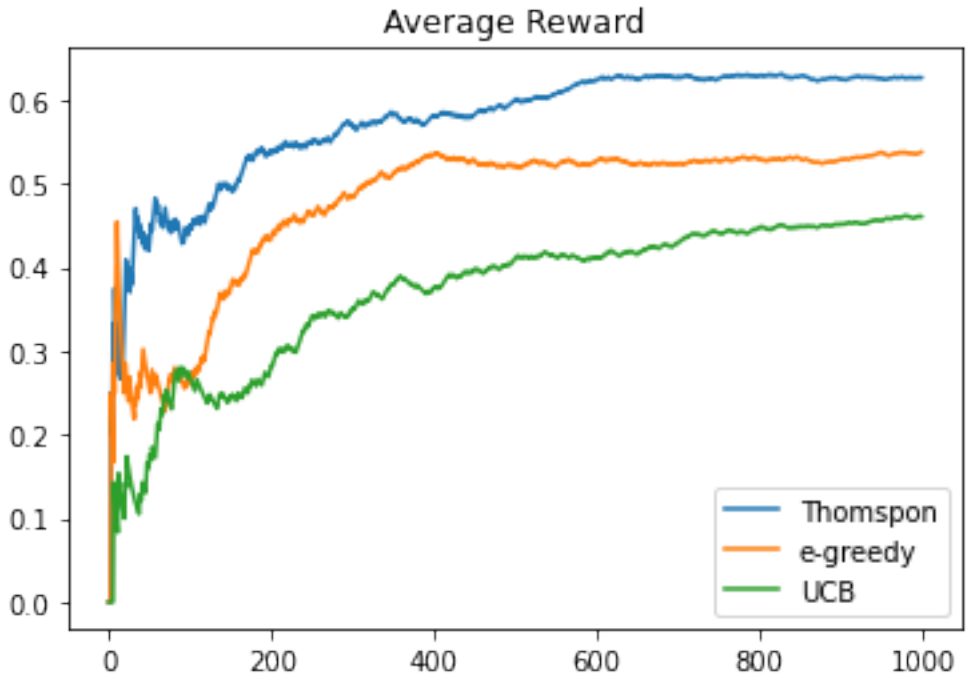

Figure 1: Average Reward vs Time plot for the algorithms

3 Part-2: Markov Decision Problems

A general MDP is characterized by the following elements:

- S=$\{s_1, s_2, s_3, ..., s_n\}$: Set of states
- A=$\{a_1, a_2, a_3, ..., a_k\}$: Set of actions
- $T(s, a, s')$: Probability of reaching s' by starting at s and taking action a
- $R(s, a, s')$: Reward for reaching s' by starting at s and taking action a
- γ: Discount factor

Apart from the ones given above, we also use a few other parameters which help us to formulate solutions for MDPs:

- Policy($\pi(s)$): $\pi(s) : S \rightarrow A$ is a map from set S to A. It defines the actions which the agent should take when it is in a particular state.
- State Values for policy π: $V_\pi(s)$ is expressed as:

$$V_\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') \big\{ R(s, \pi(s), s') + \gamma V_\pi(s') \big\} \tag{5}$$

So, now that we have defined all the parameters required, we will look at various methods of dynamic programming we can use to solve MDPs.

1. **Value Iteration:** An assignment to implement the dynamic programming algorithm value iteration from scratch was completed. The psuedo code for the algorithm is:

   V_0 is a bounded arbitary n length vector.
   $t \leftarrow 0$

   While $V_t \neq V_{t-1}$
   For all states $s \in S$
   $V_{t+1}(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \{ R(s, a, s') + \gamma V_t(s') \}$
   $t \leftarrow t + 1$

2. **Policy Iteration:** Policy iteration involves the selection of a random policy π initially. As long as π is improvable, π will be improved to a better policy. Psuedo code for policy iteration is:

   While π is improvable: $\pi' \leftarrow$ Improve policy (π)
   $\pi \leftarrow \pi'$

4 Learnings

This project was a great introduction to Reinforcement Learning and helped me learn the foundational concepts of the same. Also, this project taught me that mathematical rigor is important for studying any area of machine learning.

References

[1] All the references and coding assignments can be found here. *https://github.com/SohamInamdar142857/WiDS-Hands-on-RL*