Reinforcement Learning

SoS Report

Balaji Karedla

June 2024

Contents

1 Introduction

Edit it Later!!!

As I embarked on my journey into the realm of machine learning, one area that particularly captured my interest was reinforcement learning (RL). This subset of machine learning stands out because of its unique approach to training algorithms through interaction with an environment. Unlike supervised learning, which relies on a vast dataset of labeled examples, reinforcement learning focuses on learning optimal behaviors through trial and error.

In reinforcement learning, an agent learns to make decisions by performing actions in a given environment to achieve a specific goal. To draw a simple analogy, consider playing a video game. Here, the agent represents the player, and the environment corresponds to the game world. The player's objective is to make moves that score points or advance levels. Through continuous interaction with the game, the player gradually learns which actions yield the highest scores or successfully complete the levels.

This learning process is driven by the concept of rewards and punishments. The agent receives feedback from the environment in the form of rewards (positive feedback) or penalties (negative feedback) based on its actions. Over time, the agent aims to maximize its cumulative reward by identifying and executing actions that yield the most favorable outcomes. This method of learning closely mimics the way humans and animals learn from their surroundings, making it a compelling area of study.

Reinforcement learning has already demonstrated its potential in various domains, from game playing (such as AlphaGo and OpenAI's Dota 2 bot) to robotics, autonomous driving, and even financial trading. Its ability to adapt and improve through interaction makes it a powerful tool for tackling complex decision-making problems.

As I continue to explore this fascinating field, I am eager to delve deeper into the algorithms and techniques that drive reinforcement learning. Understanding these foundational concepts is the first step towards leveraging RL to develop intelligent systems capable of making informed and autonomous decisions.

2 Multi-Armed Bandits (MAB)

2.1 What is a Multi-Armed Bandit??

Multi-Armed Bandits are common in casinos and are one of the easily modelled games in Reinforcement Learning. The name "Multi-Armed Bandit" comes from the slot machines in casinos, also known as one-armed bandits. The slot machines have a lever (arm) the player pulls to play the game. The player can choose from multiple slot machines with a different reward distribution. The goal is to maximize the total reward by selecting the slot machine with the highest expected reward. The player faces a trade-off between exploring different slot machines to learn their reward distributions and exploiting the best-known slot machine to maximize the reward. This trade-off is known as the exploration-exploitation dilemma.

The Multi-Armed Bandit I studied is a simple version of the Reinforcement Learning problem, where an agent interacts with an environment by selecting one of the available actions (arms) at each time step.

I chose a 10-armed bandit from [1], where the mean of the rewards given by the bandit was

Figure 1: A Slot Machine or an Armed Bandit

randomly selected over a constant distribution over a $[-2, 2)$ range. The bandits give a reward chosen from a normal distribution with a mean equal to the mean of the bandit and a standard deviation of 1. The agent aims to maximize the total reward by selecting the bandit with the highest expected reward.

2.2 Agent Implementation

I implemented the agent using the different algorithms given in [1]. The implementation in all of these algorithms included storing an array of action values, which are estimates of the expected rewards of each bandit. The agent selects the action with the highest action value to *exploit*, the best-known bandit, or selects a random action to *explore* other bandits. The agent updates the action values based on the rewards received from the environment.

The updation of the action values is different across different algorithms.

2.3 Algorithms

2.3.1 Greedy Algorithm

The Greedy Algorithm is the simplest Algorithm for solving the Multi-Armed Bandit problem. The agent keeps track of the action values for each bandit and selects the bandit with the highest action value at each time step. The agent *exploits* the best-known bandit by choosing the action with the highest action value.

The action values of the bandits, represented by q_*, are estimated by the agent using the sample average method to get an estimate, represented by Q_t. The action value of a bandit a at time step t is updated using the following formula:

$$q_* \doteq \mathbb{E}[R_t | A_t = a]$$

$$Q_t(a) \doteq \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

where R_t is the reward received by the agent at time step t, A_t is the action selected by the agent at time step t, and $\mathbb{1}_{A_i=a}$ is an indicator function that is 1 if $A_i = a$ and 0 otherwise.

In the greedy algorithm, the agent always selects the bandit with the highest action value. So,

$$A_t \doteq \arg\max_a Q_t(a)$$

This definition of Q_t lets us define it recursively as follows:

$$Q_{t+1} = Q_t + \frac{1}{n}[R_t - Q_t]$$

where n is the number of times the action a has been selected.

2.3.2 Exploration-Exploitation Dilemma

The Greedy Algorithm has a drawback in that it always selects the bandit with the highest action value, which can lead to suboptimal performance. The agent may not explore other bandits to learn their reward distributions, which can result in a lower total reward.

This can be overcome by choosing a random bandit every time with equal probability to explore the bandits. This ensures the agent explores all the bandits and learns their reward distributions. But the reward is highly distributed, so the agent may be unable to exploit the best bandit.

This is the Exploration-Exploitation Dilemma in the Multi-Armed Bandit problem. This can be overcome by choosing the perfect ratio of exploration and exploitation.

2.3.3 ϵ-Greedy Algorithm

The ϵ-Greedy Algorithm is a simple solution to the Exploration-Exploitation Dilemma. The agent selects the best-known bandit with probability $1 - \epsilon$ and selects a random bandit with probability ϵ. The agent *exploits* the best-known bandit with probability $1 - \epsilon$ and *explores* other bandits with probability ϵ[1].

The action value of a bandit a at time step t is updated using the following formula:

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{n}[R_t - Q_t(a)]$$

where n is the number of times the action a has been selected.

The value of ϵ is a hyperparameter that controls the exploration and exploitation trade-off. A higher value of ϵ encourages more exploration, while a lower value of ϵ encourages more exploitation.

The 2 shows the performance of the ϵ-Greedy Algorithm with different values of ϵ. The performance of the algorithm is measured by the average reward over time. The algorithm with $\epsilon = 0.1$ performs better than the algorithm with $\epsilon = 0.01$ and $\epsilon = 0$.

It is obvious that the greedy algorithm grows faster than the other algorithm at the beginning but the ϵ-greedy algorithm with $\epsilon = 0.1$ performs better than the greedy algorithm in the long run. The $\epsilon = 0.01$ algorithm is closer to the greedy algorithm, but it performs better than the greedy

[1] Assuming there are k bandits, the probability of selecting a bandit a at time step t is given by:

$$\pi(a_t = a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{k} & \text{if } a = \arg\max_a Q_t(a) \\ \frac{\epsilon}{k} & \text{otherwise} \end{cases}$$

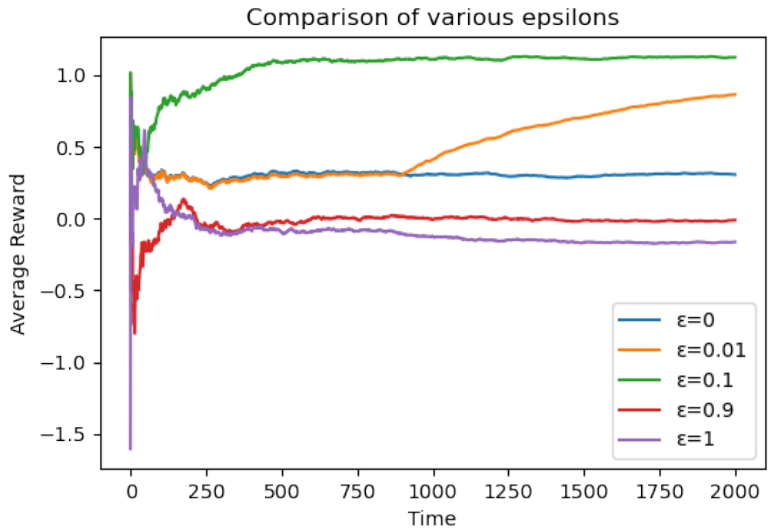

Figure 2: The ϵ-Greedy Algorithm

algorithm in the long run, it even performs better than the $\epsilon = 0.1$ algorithm in the long run, but it grows slower than the $\epsilon = 0.1$ algorithm.

Completely random selection of bandits ($\epsilon = 0$) performs the worst among all the algorithms. The agent does not exploit the best-known bandit and spends most of the time exploring other bandits, resulting in a lower total reward.

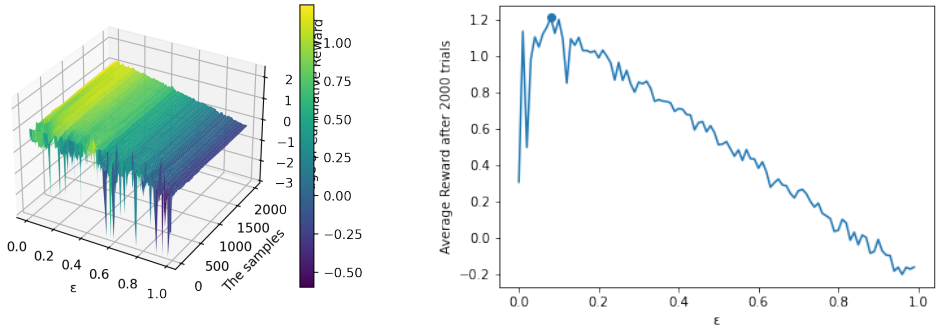

Figure 3: Comparison of epsilons in the ϵ-Greedy Algorithm

You can see that the maximum of the reward is at $\epsilon = 0.08$ and then decreases.

2.3.4 Various Other Step-sizes

The Q_t updation formula can be generalized to include a step-size parameter α to control the rate at which the action values are updated. The action value of a bandit a at time step t is updated using the following formula:

$$Q_{t+1}(a) = Q_t(a) + \alpha[R_t - Q_t(a)]$$

where α is the step-size parameter that controls the rate at which the action values are updated. A higher value of α updates the action values faster, while a lower value of α updates the action values slower.

3 Finite Markov Decision Processes

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA: MIT Press, 2018. ISBN: 978-0262039246.