SOC Week 1

Anirudha Shinde

June 5, 2024

1 Introduction

In this week I learned about some fundamental concepts in Reinforcement Learning like Markov Decision Process(MDP) and Dynamic Programming. Also gone through problem like Multi-Arm Bandits and Cart-Pole balancing.

2 Topics I learned

2.1 Multi-Arm Bandits

It is a problem in which a agent could choose multiple actions and each action has some reward and the reward is in a probability distribution. Our goal is to maximize this reward over the number of steps we are interested in. But the agent is unaware of how the probability distribution is of each action. Now i would define the problem in a bit detail.

1. **Arms (Actions):** $A = \{a_1, a_2, ..., a_k\}$
   - Each arm a_i represent a different action the agent can take.
   - k is the total number of arms.
2. **Rewards:** $R_i(t)$
   - $R_i(t)$ is the reward when a_i is played at time t.
   - The reward distribution for each arm is unknown and may differ from arm to arm.
3. **Objective:**
   - Maximize the cumulative reward over a sequence of plays.(time steps)
   - The agent needs to balance Exploration and Exploitation.

I will discuss about two ways to solve the problem which are ϵ-Greedy and Upper Confidence Bound.

Return Function: $q_*(a) = \mathbb{E}[R_t|A_t = a]$

Average reward for particular action: $Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i}{\sum_{i=1}^{t-1} 1}$

1. ϵ **- Greedy Strategy**

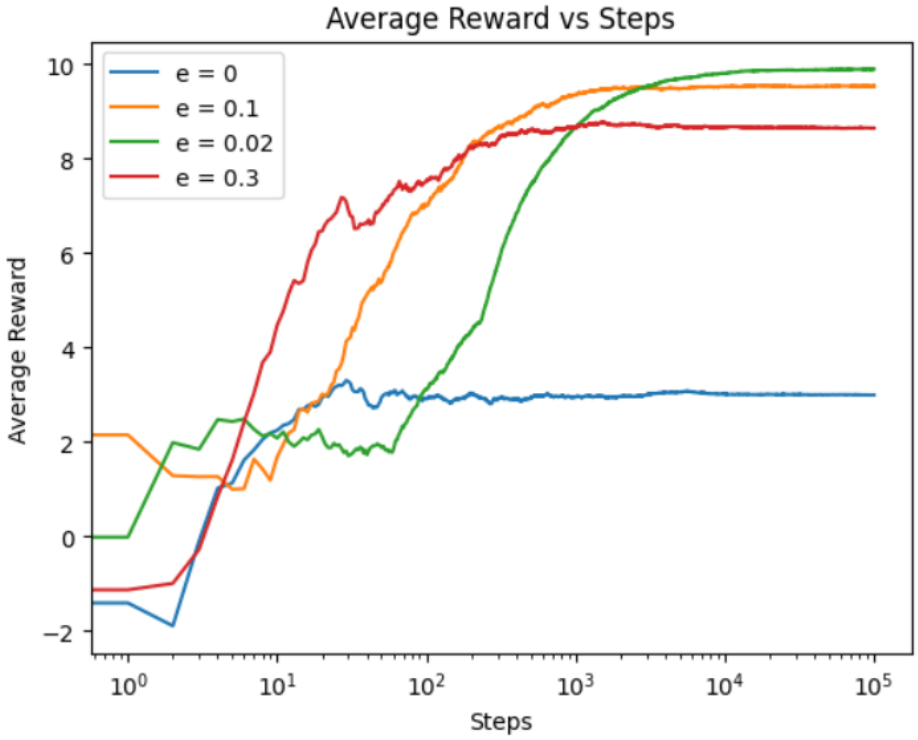
   In this approach we define a ϵ which is the probability that the action taken should be greedy or exploratory.

   If we end up with a greedy action it will be:

$$A_t = \underset{a}{argmax}\, Q_t(a)$$

   Now we update our return function in a recursive manner like given below:

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



2. **Upper Confidence Bound**

   In the UCB our single goal is to maximize the function which is derived from hoffdings inequality. 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. One effective way of doing this is to select actions according to

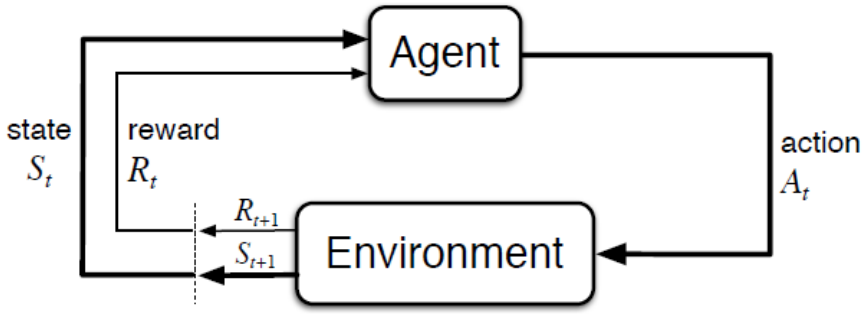
$$A_t = \underset{a}{argmax}[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}]$$

2.2 Markov Decision Process

Markov Decision process is a mathematical framework used for forming model and solving decision making problems which are uncertain and dependent on both Agents action and environments response in a probabilistic way.Let me define some basic terms in MDP:

1. **States (S)**: A finite set of states that represent all possible situations in which the decision-maker could be.
2. **Actions (A)**: A finite set of actions available to the decision-maker. Each action can lead to different outcomes or transitions.
3. **Transition Probability (P)**: The probability of transitioning from state s to state $s^{'}$ given action a , denoted as $P(s^{'}|s,a)$
4. **Reward (R)**: The immediate reward received after transitioning from state s to state $s^{'}$ given action a , denoted as $R(s^{'}|s,a)$
5. **Policy (π)**: A strategy or policy that specifies the action to be taken in each state.

Basic working between Agent and Environment is best described in this figure.



In Reinforcement Learning, the purpose of the agent is to maximize the expected return, where the return is denoted by G_t. This could be written in a mathematical form which recursive and which helps in computing returns from reward sequences.

$$G_t = \sum_{k=0}^{T} \gamma^k . R_{t+k+1}$$

$$G_t = R_{t+1} + \gamma . G_{t+1}$$

Value Function of a state s under the policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π.

- **State-Value Function:** $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$

Similarly, we define the value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$ as the expected return starting from s, taking the action a, and following the policy π.

- **Action-Value Function:** $q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a]$

The Bellman equations provide recursive definitions for the value functions used in Markov Decision Processes (MDPs). There are two primary value functions: the state value function (v) and the action value function (q). Below are the Bellman equations for each.

- **Bellman Equation for the State Value Function**

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s,r} P(s^{'}, r|s, a).[r + \gamma v_\pi(s^{'})]$$

- **Bellman Equation for the Action Value Function**

$$q_\pi(s, a) = \sum_{s^{'},r} p(s^{'}, r|s, a)[r + \gamma \sum_a \pi(a|s).q_\pi(s^{'}, a^{'})]$$

- **Bellman Optimality Equation for the State Value Function**

$$v_*(s) = \max_a \sum_{s^{'},r} p(s^{'}, r|s, a)[r + \gamma v_*(s^{'})]$$

- **Bellman Optimality Equation for the Action Value Function**

$$q_*(s, a) = \sum_{s^{'},r} p(s^{'}, r|s, a)[r + \gamma \max_{a^{'}} q_*(s^{'}, a^{'})]$$

2.3 Dynamic Programming

Dynamic Programming is method of solving a problem which needs decision making in the process, and this process assumes that agent has the complete knowledges of the environment.

1. **Policy Evaluation**
   In this technique our goal is to find expected return $v_\pi(s)$ from starting state to the terminal state for a given policy. We do this by applying the Bellman equation for value function.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s,r} P(s^{'}|s, a).[r + \gamma v_\pi(s^{'})]$$

2. **Policy Improvement**
   We now know the value of a policy π, we wish to improve our current policy so that we could maximize the expected return. So we find a new policy $\pi^{'}$that is at least as good as the previous policy π, this is done by the equation given below:

$$\pi^{'}(s) = \arg\max_a \sum_{s^{'},r} p(s^{'}, r|s, a)[r + \gamma v_\pi(s^{'})]$$

3. **Policy Iteration**
   This is an iterative process that alternates between policy evaluation and policy improvement until the policy converges to the optimal policy.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

   steps:

   1 **Policy Evaluation:** Compute $v_\pi(s)$ for current policy π.

2 **Policy Improvement:** Use v_π to find an improved policy π'.

3 Repeat until π no longer changes.

4. **Value Iteration**
   An alternative to policy iteration, that combines policy evaluation and policy improvement into a single step. 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