Hands-On Reinforcement Learning

Aadish Sethiya
Computer Science Engineering
Indian Institute of Technology
aadi.sethiya@gmail.com

***Abstract*—This document gives an insight into the world of reinforcement learning a it's various aspects and components. We dive deep into the theory of multi-armed bandits and Markov Decision Processes. We also deal with the concepts of Value Functions and Value and Policy Iteration.**

I. Introduction

Reinforcement Learning is a computational approach to learn from interaction with environtment. It is just like how us as human beings learn from our experiences and implement then in similar future jobs.It consists of 2 components, trial and error search, delayed reward and exploration vs exploitation tradeoff. The two main elements of reinforcement learning is agent and environment.

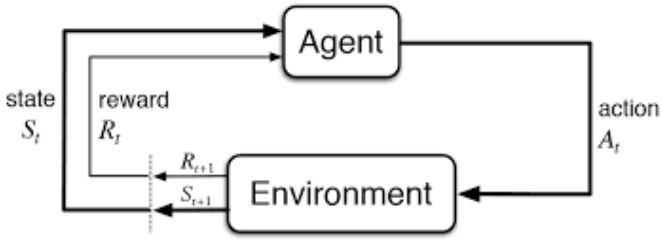

Fig. 1. Interaction between agent and environment

The subelements of reinforcement are:

- **Policy**: mapping from states to actions.
- **Reward**: Given by the environment at each time step.
- **Value Function**: Total amount of reward an agent can expect to accumulate over the future, starting from that state.

II. Multi-Armed Bandits

Reinforcement learning uses with evaluative feedback approach to train a model. Evaluative feedback indicates how good the action taken was. We use k-armed bandits to learn about this evaluative feedback in a simplified setting. In a k-armed bandit setting we repeatedly make choices among k given actions. After each action we receive a reward. Out main objective is to maximize the total expected reward overtime. There are 4 parameters that we deal with when working with k-armed bandit. These are:

- $\boldsymbol{A_t}$: Action taken at time t.
- $\boldsymbol{R_t}$: Reward of action $\boldsymbol{A_t}$
- **q(a)**: Value of the action a.
- $\boldsymbol{Q_t(a)}$: Estimated value of the action a at time t.

The value of q(a) is given by the following equation:

$$q(a) = \mathbb{E}[R_t|A_t = a]$$

A. Greedy and Non-Greedy Actions

At each time step, for taking an action we have two choices to make namely **greedy** action and **non-greedy** action. A greedy action is an action with highest q value. This way we do get maximum reward available to us at each step however this leads to exploitation as there are chances that the value of some other action might exceed the value of greedy action if it is explored more. Such a action with lower current estimate is called non-greedy action. We always use a perfect blend of greedy and non greedy actions to pick an action at each step. A way to encourage exploration is to select initial estimates to a higher number so that unexplored states ave a higher value than explored ones arnd are selected more often.

B. Action Value Methods

Action-value methods are the methods used to compute $Q_t(a)$. One way of doing so is by averaging over the rewards received over time. For a single action, the estimate Q_n of this action value after it has been selected $n-1$ times is:

$$Q_n = \frac{R_1 + R_2 + ... + R_{n-1}}{n-1}$$

In order to reduce the storage capacity by using the previous computed value, this equation can easiy be broken down to

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

As denominator goes to infinity, by the law of large numbers $Q_t(a)$ converges to $q(a)$. Then for a greedy action we pick $A_t = \arg\max_a Q_t(a)$

The method discussed above works for stationary problems and not non-stationary problems. In a non-stationary problem, the reward proobabilities change over time hence we want to give more weightage to the most recent rewards. We can achieve this by keeping a constant step size to update Q.

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n]$$

C. Epsilon-Greedy and UCB

1) Epsilon-Greedy: We have earlier discussed about the two choice available to us to select an action at each time step namely greedy actions and non-greedy actions. Epsilon greedy approach uses a mixture of these 2 actions to blend

exploitation and exploration together. In this method we define a variable epsilon such that $0 < \epsilon < 1$. We continue to behave greedily except for a small proportion ϵ of the actions where we pick a random action. The picking of a random action ensures we are are exploring all the actions available to us.

2) UCB: This method is called upper confidence bound ad this method provides us with an additional upper bound to select an action based on their potential of being optimal. At at instant of time t, the action A_t selected is given by:

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

Where $N_t(a)$ is the number of times action a has been selected prior ro time t and $c > 0$ is the confidence level that controls the degree of exploration. The square-root term is a measure of the uncertainty or variance in the estimate of a's value. Each time a is selected the uncertainty is reduced and each time any other action is selected, $\ln t$ increases but not $N_t(a)$ so the uncertainty increases.

The following graphs compare both epsilong greedy and UCB algorithms where the value of confidence level for UCB is 3 and the value of ϵ for epsilon greedy algorithm is 0.1.

Fig. 2. Avg Reward vs Time

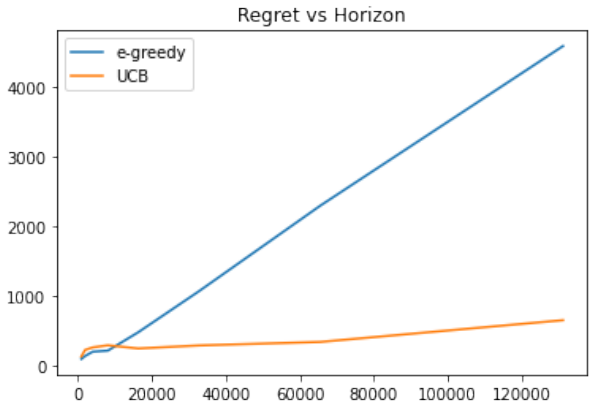

Fig. 3. Regret vs Horizon

III. Markov Decision Process

A Markov decision process also known as MDP has 5 major elements:

- **S**: A finite set of states that describes the environment.
- **A**: A set of finite actions that describe the action that ca be taken by the agent.
- **T**: A transition function. For s,s' ∈ S, a ∈ A: T(s,a,s') is the probability of reaching s' by starting at s and taking action a.
- **R**: A reward function. For s,s' ∈ S, a ∈ A: R(s,a,s') is the reward for reaching s' by starting at s and taking action a. Negative rewards indicate bad results while positive rewards indicate good results.
- γ: A discount factor. It tells how important future rewards are to the current state. It is between 0 and 1.

The agent at each point of time interacts with the environment and picks a suitable action to return to the same state or to go to the next step. We look at the preceding history to select an optimal action. In other words we follow a policy π: S $\rightarrow$.

A. Value Functions

The goal of a the agent is to maximize cumulative knowledge about how to achieve the task. The expected return we want to maximize can be defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

We can break down the above equation in the following way:

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

Value function defines the expected return in a state. The value function of a state s under a policy π, denoted by $V^\pi(s)$, is the expected return starting in s and following π thereafter.

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

Similarly action-value function $Q^\pi(s)$ is defined as

$$Q^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$V^\pi(s)$ and $Q^\pi(s)$ can be determined by experience. If we maintain an average reward received for each state, this converges to $V^\pi(s)$.

Value function recursively be written as:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} T(s,a,s')[R(s,a,s') + \gamma V^\pi(s')]$$

This equation is known as **Bellman equation**, It expresses the relationship between the value of a state and the values of its successor states.

B. Optimality Equations

Among all the policies for a given MDP, there must be an optimal policy π^*. This optimal policy has following characteristics:

- $\pi^* \geq \pi$ iif $V^{\pi^*}(s) \geq V^{\pi}(s)\ \forall s\ \in\ S$
- The optimal functions are called V^* and Q^*

The equation for V^* is given as:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

The corresponding optimal policy is given by:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

C. Policy Evaluation

In this we input a MDP and a policy π and get state-value function V^π as an output. To compute V^π iteratively we choose an arbitary value of the initial state and each successive approximation is obtained by using the Bellman equation for V^π as an update rule. This algorithm is called **iterative policy evaluation**. The pseudocode for this algorithm is as follows:

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Fig. 4. Iterative policy evaluation

D. Policy Improvement

For a state s we would like to know if we should select an action $a \neq \pi(s)$, a better raction than the one currenntly advised by out policy. One way to do this is to select a and thereafter following the existing policy π:

$$Q^\pi(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi(s'))]$$

If this quantity is greater than $V^\pi(s)$, it means it's better to take action a then follow π than to follow π all the time. The natural extension to this is to consider all states and all actions. In a greedy policy π', we select at each step the action that appears best according to $Q^\pi(s)$.

$$\pi'(s) = \arg\max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\pi * (s')]$$

E. Policy Iteration

This method consists of following steps:

1) Take policy π_n
2) Compute V^{π_n}
3) Use V^{π_n} to compute better policy π_{n+1}
4) Repeat until convergence

Each policy is guaranteed to be strict improvement of the previous one. This process converges to an optimal policy and optimal value function in a number of steps. The pseudocode for policy iteration is as follows:

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
     $\Delta \leftarrow 0$
     For each $s \in \mathcal{S}$:
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
     $a \leftarrow \pi(s)$
     $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
     If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Fig. 5. Policy Iteration

F. Value Iteration

Every iteration of policy iteration involves policy evaluation to compute V. Convergence to V^π occurs only in the limit. Value iteration is just policy iteration stopped after one sweep. We are turning Bellman optimality equation into an update rule where value iteration update is identical to policy evaluation update except here we max over all actions. Each intermediate construct of V doesn't correspond to any policy. The pseudocode for value iteration is as follows:

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Fig. 6. Value Iteration